(19) OFICIUL DE STAT
PENTRU INVENŢII ŞI MĂRCI
Bucureşti

ROMÂNIA

(11) **RO 123609 B1**

(51) Int.Cl.
G06T 1/00 (2006.01),
G06T 11/00 (2006.01),
G06T 13/00 (2006.01),
G06T 15/00 (2006.01),
G06T 19/00 (2011.01),
G06F 3/048 (2006.01)

(12) # BREVET DE INVENŢIE

(21) Nr. cerere: **a 2003 00414**

(22) Data de depozit: **15.05.2003**

(45) Data publicării menţiunii acordării brevetului: **30.07.2014** BOPI nr. **7/2014**

(30) Prioritate:
27.03.2003 US 10/401.717

(73) Titular:
• **MICROSOFT CORPORATION**,
ONE MICROSOFT WAY, REDMOND, WA, US

(72) Inventatori:
• **JOSEPH S. BEDA**,
3819 DENSMORE AVENUE N., SEATTLE, WA, US;
• **KEVIN T. GALLO**,
19235 222nd WAY N.E., WOODINVILLE, WA, US;
• **ADAM M. SMITH**,
12310 N.E. 92nd STREET 205, KIRKLAND, WA, US;
• **GILMAN K. WONG**,
10509 176th PLACE N.E., REDMOND, WA, US;
• **SRIRAM SUBRAMANIAN**,
6209 104th AVENUE N.E., KIRKLAND, WA, US

(74) Mandatar:
ROMINVENT S.A.,
STR. ERMIL PANGRATTI NR.35, SECTOR 1, BUCUREŞTI

(56) Documente din stadiul tehnicii:
US 6215495 B1

(54) # SISTEM PENTRU PROCESAREA INFORMAŢIILOR GRAFICE ŞI A ALTOR INFORMAŢII VIDEO PENTRU AFIŞAREA PE SISTEME DE CALCULATOARE

(57) **Rezumat:**

Invenţia se referă la un model obiect de elemente, precum şi la un limbaj de marcare de grafică vectorială, pentru utilizarea acestui model obiect de elemente într-o manieră care permite dezvoltatorilor de coduri de programe să se interfaţeze într-o manieră consistentă cu o structură de date a grafului scenic, pentru a produce grafică. Modelul obiect al elementelor de grafică vectorială corespunde, în general, elementelor de conformaţie, precum şi altor elemente ce includ elemente de imagine şi video, care se găsesc în corelaţie cu un model obiect de graf scenic al grafului de decor. Marcatorul poate fi analizat gramatical în date ce includ elemente dintr-un arbore de elemente, care este tradus în obiecte ale unei structuri de date a grafului scenic. Alt marcator poate fi tradus direct în date şi apeluri care creează obiectele grafului scenic. Limbajul de marcare asigură căi distincte de a descrie un element, incluzând un format simplu al şirului sau o sintaxă complexă de proprietăţi care pot fi menţionate, permiţând refolosirea în alte locaţii din marcator.

Revendicări: 27
Figuri: 27

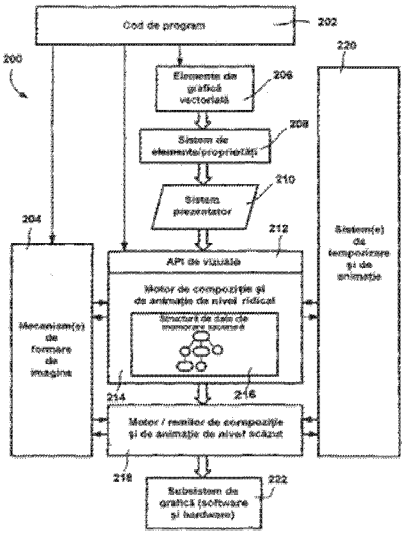

Fig. 2

Examinator: **ing. DUMITRU DANIELA**

RO 123609 B1

Prezenta invenție se referă la un sistem pentru procesarea informațiilor grafice şi a altor informații video, pentru afişarea pe sisteme de calculatoare, fiind în legătură cu următoarele cereri de brevete de invenție, dependente, din Statele Unite: **US 2003/0076328**, intitulată "Sistem şi metodă de procesare a graficii de nivel multiplu"; **US 2003/0132937**, intitulată "Parametrizare generică pentru un graf scenic"; **US 2003/0076329**, intitulată "Structură inteligentă de memorare ascunsă a datelor pentru grafica de mod imediat"; fiecare depusă pe 27 iunie 2002; precum şi cererea de brevet de invenție din Statele Unite, intitulată "Interfețe grafice vizuale şi scenice" (Registru mandatar **US 2003/0693673**).

Invenția se referă, în general, la sistemele de calculatoare şi, în mod mai particular, la prelucrarea informațiilor grafice şi a altor informații video, pentru afişarea pe sistemele de calculatoare.

Limitele modelului tradițional de mod nemijlocit al graficii de accesare pe sistemele de calculatoare au fost atinse, în parte, deoarece vitezele memoriei şi ale magistralei nu au ținut pasul cu progresele din procesoarele principale şi/sau din procesoarele grafice. În general, modelul curent (de exemplu, WM_PAINT) pentru pregătirea unui cadru necesită prea multă procesare a datelor, pentru a ține pasul cu viteza de împrospătare a hardware-lui, atunci când sunt dorite efecte de grafică complexă. Drept rezultat, dacă efectele graficii complexe sunt probate, împreună cu modelele de grafică convențională, în loc de a completa modificările care rezultă din efectele vizuale, percepute în timp, pentru cadrul următor, schimbările pot fi adăugate peste cadre diferite, determinând rezultate care nu sunt dorite din punct de vedere vizual şi perceptibil.

Un model nou, pentru comanda extragerii graficii, este descris în cererile de brevet de invenție din Statele Unite, menționate mai sus, cu numerele seriale **US 2003/0076328**; **US 2003/0132937** şi **US 2003/0076328**. Acest model nou asigură un număr de îmbunătățiri semnificative în tehnologia de procesare a graficii. De exemplu, brevetul **US 2003/0076328** este orientat, în general, către un sistem şi către o metodă de procesare a graficii de nivel multiplu, la care o componentă de nivel mai ridicat (de exemplu, al unui sistem de operare) execută aspecte computaționale intense de construire a unui graf scenic, de actualizare a parametrilor de animație şi de a trece în revistă structurile de date ale grafului scenic, la o viteză de funcționare relativ scăzută, pentru a trece structurile de date simplificate şi/sau comenzile de grafică la o componentă de nivel scăzut. Deoarece procesarea de nivel ridicat simplifică în mod considerabil datele, componenta de nivel scăzut poate să funcționeze la o viteză mai mare (relativ la componenta de nivel ridicat), cum ar fi o viteză care corespunde la viteza de reîmprospătare a cadrului din subsistemul de grafică, pentru a procesa datele în date constante de ieşire, pentru subsistemul de grafică. Dacă se utilizează animația, în loc de a avea o redesenare a întregii scene cu modificări, procesarea de nivel scăzut poate interpola intervalele de parametri după cum este necesar, pentru a obține valori instantanee care, atunci când sunt livrate, asigură o scenă modificată puțin, pentru fiecare cadru, asigurând animație plăcută.

Cererea de brevet **US 2003/0132937** descrie un graf scenic, parametrizat, care asigură valori variabile (animate), precum şi recipiente de grafuri parametrizate, cum ar fi acel cod de program care doreşte să deseneze grafică (de exemplu, un program de aplicație sau o componentă a sistemului de operare), ce poate să modifice în mod selectiv anumite aspecte ale descrierii grafului scenic, în timp ce lasă intacte alte aspecte. Codul de program poate, de asemenea, să reutilizeze porțiunile deja construite ale grafului scenic, împreună cu diferiți parametri posibili. Aşa după cum poate fi apreciat, abilitatea la modificarea uşoară a apariției elementelor afişate prin parametrizare şi/sau reutilizarea părților existente ale unui graf scenic, asigură, în mod eficient, câştiguri substanțiale în procesarea graficii generale.

Cererea de brevet **US 2003/0076329** descrie, în general, o structură de memorare ascunsă a datelor, precum şi mecanisme specificate pentru stocarea informaţiilor cu privire la vizual, prin obiecte şi date dintr-un graf scenic. Structura datelor este în general asociată cu mecanisme care comandă în mod inteligent felul cum informaţia vizuală din acesta este populată şi utilizată. De exemplu, cu excepţia cerută în mod specific de către programul aplicaţiei, majoritatea informaţiei stocate în structura de date nu are nicio referinţă externă asupra ei, care permite acestei informaţii să fie optimizată sau de altfel să fie procesată. Aşa după cum poate fi apreciat, aceasta asigură eficienţă şi conservare de resurse, de exemplu, datele din structura "cache" (memorie ascunsă) de date pot fi procesate într-un format diferit, care este mai compact, şi/sau reduce nevoia de procesare ulterioară, repetată, cum ar fi un "bitmap" sau alte rezultate ale post procesării.

În timp ce îmbunătăţirile de mai sus asigură beneficii substanţiale în tehnologia procesării graficii, este încă necesar de a se găsi o cale pentru programe, pentru a utiliza efectiv acest model îmbunătăţit de grafică, precum şi alte din îmbunătăţirile acesteia, descrise într-o manieră directă. Ceea ce este necesar este o cale directă, încă intuitivă pentru programe, spre a obţine avantaje asupra multor trăsături şi asupra capabilităţilor de procesare grafică, asigurate de către modelul îmbunătăţit de grafică şi, prin aceasta, spre a se obţine grafică complexă, într-un mod eficient.

Un alt exemplu de management alternativ al unei scene 3D este descris în brevetul **US 6215495 B1**, Brad Grantham, 2011. Se creează o scenă 3D originală, descrisă în format VRML. Sunt prezentate atât componentele scenei (geometrie, materiale, lumini, imagini, imagini în mişcare şi sunete), cât şi relaţiile între acestea, statice sau dinamice. Un număr de obiecte, care reprezintă blocurile de bază pentru construirea unui graf scenic, sunt stocate într-o memorie. Obiectele constau în geometrie, stare grafică, ierarhie de transformare şi informaţii audio. Scenele în format VRML sunt transmise către o interfaţă de tip API, care apelează diverse motoare pentru a modifica graful scenic. La aceasta contribuie şi un parser, pentru a permite fişierelor VRML să poată fi rulate pe orice calculator. Interfaţa API este structurată ca o colecţie de clase ierarhizate, reprezentând aparenţa, materialul, textura, transformare de text, geometrie, culoare, coordonate, sunet, formă. Motoarele de compunere folosite sunt: un motor de morfologie geometrică, un motor de interpolare şi un motor de scriere. Între primitivele din setul geometric, sunt incluse: set de puncte, set de linii, set de poligoane, set de coordonate, set de culori, set de coordonate de textură, set de indici de integrare, primitive de tip con şi de tip cilindru.

Graful scenic este reprezentat ca un graf aciclic de noduri. Ierarhia claselor cuprinde şi un grup de noduri de tip: nod de transformare, nod de comutare şi un fundal pentru definirea luminii şi ceţii.

Problema tehnică, pe care o rezolvă invenţia, constă în producerea de grafică pe computer, prin accesarea unui model obiect de elemente, într-un mod care să permită interfaţarea, în permanenţă, cu o structură de date, a grafului scenic.

Sistemul pentru procesarea informaţiilor grafice şi a altor informaţii video, pentru afişarea pe sisteme de calculatoare, cuprinde:

- un motor de compunere şi de animaţie de nivel superior şi un motor de nivel inferior, motorul de nivel superior fiind instanţiat pe o bază per-aplicaţie, iar motorul de nivel inferior deservind cereri de la mai multe aplicaţii;

- un limbaj marcator, acesta cuprinzând instrucţiuni grafice, care cuprind un format şir şi o notaţie de obiect, notaţia de obiect cuprinzând elemente grafice de la o clasă de elemente grafice;

- un model obiect grafic, cuprinzând:

a. un obiect vizual de clasă de bază, care este un container pentru conținut grafic, ce asigură funcționalitate de bază, pentru alte tipuri vizuale şi din care derivă alte tipuri vizuale;

b. un obiect vizual de tip container, care este un container pentru elemente vizuale şi care poate conține alte obiecte vizuale de tip container;

c. un obiect vizual de tip desen care este un container pentru conținut grafic, şi

d. o clasă de elemente grafice, clasa de elemente cuprinzând o clasă de formă, o clasă de imagine, o clasă video şi o clasă Canavas, iar clasa de elemente fiind integrată cu un sistem de proprietăți generale:

- un convertor de tip, care este configurat să convertească o instrucțiune grafică în format şir la un obiect de interfață pentru programare de aplicație (API) vizual;

- un parser/translator, care este configurat să:

a. interpreteze instrucțiunile grafice, acestea cuprinzând apeluri de cod directe, apeluri de cod de model obiect şi instrucțiuni grafice scrise utilizând limbajul marcator,

b. acceseze convertorul de tip, acesta fiind configurat să convertească o instrucțiune grafică în format şir într-un obiect API vizual, şi să

c. interpreteze codul marcator şi, la interpretarea codului marcator, să adauge elemente din clasa elementelor grafice la un element arbore;

d. un sistem de prezentare, acesta fiind configurat sa traducă arborii de elemente grafice în apeluri la un API vizual; un API vizual, API-ul vizual fiind configurat să:

a. interfațeze cu sistemul de prezentare, să interfațeze cu parserul-translator şi să interfațeze cu apelurile de cod directe de la limbajele de programare, şi

b. ca răspuns la cererile sistemului de prezentare, parserul-translator creează obiecte de scenă în cadrul unui graf scenic; şi

c. o interfață de afişare operabilă să faciliteze afişarea obiectelor grafice în cadrul grafului scenic.

Sistemul conform invenției are elementele din modelul obiect de elemente corelate cu obiectele din modelul obiect al grafului scenic.

În sistemul conform invenției, marcatorul include textul în-linie, ce include un şir care defineşte o proprietate a elementului, iar translatorul comunică cu un convertor de tip, pentru a converti şirul într-o proprietate a obiectului.

În sistemul conform invenției, marcatorul include textul în-linie ce conține sintaxa de proprietate, sintaxa de proprietate specificând atribute multiple ale obiectelor grafice vectoriale.

Sistemul conform invenției este caracterizat prin aceea că textul în-linie este identificat cu o referință care se referă la o altă locație din marcator.

Sistemul conform invenției este caracterizat prin aceea că textul în-linie este identificat cu o referință care se referă la un fişier.

Sistemul conform invenției este caracterizat prin aceea că textul în-linie este identificat cu o referință care corespunde la un fişier care poate fi descărcat dintr-o locație aflată la distantă, într-o rețea.

În sistemul conform invenției, marcatorul include textul în-linie ce cuprinde sintaxa de proprietate complexă, corespunzând unei resurse grafice.

Sistemul conform invenției este caracterizat prin aceea că resursa grafică descrie un obiect pensulă vizual, parserul/traducătorul asigurând datele nivelului resursă pentru "comunicarea directă cu stratul API vizual, pentru a crea un obiect vopsea vizual, ce corespunde elementului descris de către sintaxa de proprietate complexă.

În sistemul conform invenției, datele nivelului resursă sunt identificate cu o referință care se referă la o altă locație din marcator.

De asemenea, datele nivelului resursă sunt identificate cu o referință care se referă la un fişier.

În sistemul conform invenției, datele nivelului resursă sunt identificate cu o referință care se referă la un fişier care poate fi descărcat dintr-o locație aflată la distantă, într-o rețea.

Sistemul conform invenției este caracterizat prin aceea că unul dintre elementele modelului obiect grafic cuprinde un element imagine.

Sistemul conform invenției este caracterizat prin aceea că unul dintre elementele grafice cuprinde un element polilinie. De asemnea, unul dintre elementele grafice cuprinde un element poligon sau un element traiectorie, sau un element linie sau un element elipsă sau un element cerc.

Sistemul conform invenției este caracterizat prin aceea că unul dintre elementele grafice include cel puțin date despre proprietatea de umplere şi/sau date despre proprietatea de mişcare a pensulei şi/sau date despre proprietatea de decupare şi/sau date despre proprietatea de transformare şi/sau date despre efect şi/sau date despre opacitate şi/sau date despre modul de îmbinare.

În sistemul conform invenției, translatorul solicită instanțierea a cel puțin unui constructor, pentru a crea obiecte de grafică.

Avantajele care decurg din aplicarea invenției sunt menționate în continuare. Astfel:

Sistemul de grafică vectorială poate astfel să programeze la un nivel element, la care fiecare dintre conformațiile de desenare sunt reprezentate ca un element la acelaşi nivel ca şi restul elementelor programabile dintr-o pagină/ecran, permițând interacțiunea cu sistemul de prezentatori, cu evenimente şi cu proprietăți.

Sistemul de grafică vectorială prevede, de asemenea, un mecanism pentru programarea la un nivel resursă, prin care designerii scenici pot în fond să scurtcircuiteze arborele de elemente şi sistemul de prezentatori, şi să programeze direct la stratul API al vizualului, care se interfațează cu structura de date a grafului scenic. Acesta asigură o cale mai eficientă şi mai uşoară, pentru a obține obiectul potrivit, cu toate că se pierde ceva din programabilitatea nivelului element. La o implementare, dacă este programată o umplere de tipul "pensulă vizual", analizorul gramatical poate să apeleze direct stratul API cu datele nivelului resursă, pentru a crea un obiect vopsea al vizualului corespunzător (care este şi o corelare dintre modelul obiect element şi modelul obiect graf scenic). La acest sistem bistratificat, grafica vectorială la nivel de element este analizată gramatical în elementele create, care necesită o traducere ulterioară în obiecte, în timp ce grafica vectorială la nivel de resursă este analizată gramatical şi stocată direct, într-un mod eficient. În acelaşi timp, datele nivelului resursă sau obiectele create din acesta pot fi completate de către elemente şi de către o parte din arborele de elemente. În acest scop, pot fi denumite elementele ce includ elemente de vopsea pentru vizual. Proiectantul de scenă are astfel abilitatea de a aprecia eficient programabilitatea, atât cât este necesar.

Alte foloase şi avantaje ale invenției vor deveni evidente din următoarea descriere detaliată, dacă se face legătura cu desenele, în care:

- fig. 1 este o schemă bloc, ce reprezintă un sistem tipic de computer în care poate fi încorporată prezenta invenție;
- fig. 2 este o schemă bloc, ce reprezintă, în general, o arhitectură de nivel de grafică, în care poate fi încorporată prezenta invenție;
- fig. 3 este o reprezentare a unui graf scenic de vizuale, precum şi componentele asociate, pentru procesarea grafului scenic, cum ar fi prin parcurgerea grafului de decor, pentru a asigura comenzile graficii, precum şi alte date, în conformitate cu un aspect al invenției de față;

- fig. 4 este o reprezentare a unui graf scenic de vizuale de validare, de vizuale de desenare şi de primitive asociate de desenare, create în conformitate cu un aspect al prezentei invenţii;

- fig. 5 este o reprezentare a unei clase de imagini vizuale, a unui model obiect, în conformitate cu un aspect al invenţiei de faţă;

- fig. 6 este o reprezentare a altor diverse obiecte ale modelului obiect, în conformitate cu un aspect al prezentei invenţii;

- fig. 7 este o diagramă ce reprezintă transformarea unor date ale vizualului, în conformitate cu un aspect al prezentei invenţii;

- fig. 8A şi 8B sunt reprezentări ale transformărilor unor date ale vizualului într-o scară geometrică şi, respectiv, într-o scară neuniformă, în conformitate cu un aspect al prezentei invenţii;

- fig. 9A...9C sunt scheme bloc ale obiectelor vizuale de suprafaţă, precum şi ale altor vizuale şi componente, în conformitate cu un aspect al invenţiei de faţă;

- fig. 10A şi 10B sunt diagrame, ce reprezintă obiectele vizuale HWnd, în conformitate cu un aspect al prezentei invenţii;

- fig. 11 este o diagramă, ce reprezintă un obiect vizual stratificat, în conformitate cu un aspect al prezentei invenţii;

- fig. 12 este o reprezentare a claselor de geometrie ale modelului obiect, în conformitate cu un aspect al invenţiei de faţă;

- fig. 13 este o reprezentare a unei structuri de geometrie a traiectoriei ("PathGeometry"), în conformitate cu un aspect al prezentei invenţii;

- fig. 14 este o reprezentare a unui graf scenic al vizualelor şi al primitivelor de desenare, ce arată o grafică de exemplu, produsă de către primitive, în conformitate cu un aspect al invenţiei de faţă;

- fig. 15 este o reprezentare a claselor de pensule din modelul obiect, în conformitate cu un aspect al invenţiei de faţă;

- fig. 16 este o reprezentare a graficii remise, ce rezultă din datele dintr-un obiect pensulă cu gradient liniar, în conformitate cu un aspect al prezentei invenţii;

- fig. 17 este o reprezentare a graficii remise, ce rezultă din datele dintr-un obiect pensulă cu gradient radial, în conformitate cu un aspect al prezentei invenţii;

- fig. 18 este o reprezentare a graficii remise, ce rezultă din a avea diverse valori de extindere, în conformitate cu un aspect al invenţiei de faţă;

- fig. 19 este o reprezentare a graficii remise, ce rezultă din a avea diverse valori ale plăcii (tile), în conformitate cu un aspect al prezentei invenţii;

- fig. 20 este o organigramă ce reprezintă într-un mod general logica pentru interpretarea unui vizual, incluzând un obiect pensulă, pentru a genera grafică, în conformitate cu un aspect al invenţiei de faţă;

- fig. 21 este o reprezentare a unei grile şi a unei grile transformate, ce rezultă din datele dintr-un obiect pensulă al vizualului, în conformitate cu un aspect al prezentei invenţii;

- fig. 22 este o reprezentare a unei grile şi a unei grile transformate, împreună cu grafica remisă, care este desenată de la un vizual, în conformitate cu un aspect al prezentei invenţii;

- fig. 23 este o reprezentare a unui obiect pensulă remis, cu nouă grile, în conformitate cu un aspect al invenţiei de faţă;

- fig. 24 este o reprezentare a claselor transformatei, ale modelului obiect, în conformitate cu un aspect al prezentei invenţii.

- fig. 25 este o reprezentare a claselor elementelor din modelul obiect element, în conformitate cu un aspect al prezentei invenţii;

- fig. 26 este o reprezentare a componentelor, pentru determinarea codului limbajului de marcare să se interfațeze cu stratul API vizual, în conformitate cu un aspect al invenției de față;

- fig. 27 este o reprezentare a decupării printr-o cale geometrică, în conformitate cu un aspect al prezentei invenții.

Fig. 1 ilustrează un exemplu de mediu potrivit de sistem de calcul **100**, pe care invenția poate fi implementată. Mediul **100** al sistemului de calcul este numai un exemplu de mediu corespunzător de calcul şi nu este avut în vedere să se sugereze nicio limitare, la fel ca la scopul utilizării sau ca la funcționalitatea invenției. Niciun mediu, **100**, de calcul nu va fi interpretat ca având vreo dependență sau vreo cerință referitoare la oricare componentă sau la oricare combinație de componente ilustrate în mediul de operare tipic, **100**.

Invenția este funcțională, împreună cu numeroase alte medii sau configurații de sisteme de calcul cu scopuri generale sau cu scopuri speciale. Exemple ale binecunoscutelor sisteme de calcul, medii şi/sau configurații, care pot fi potrivite pentru utilizarea împreună cu invenția, includ, dar nu sunt limitate la acestea, calculatoare personale, calculatoare servere, dispozitive "hand-held" sau "laptop"-uri, dispozitive tip tabletă, sisteme multiprocesor, sisteme bazate pe microprocesor, incinte acționate în partea de sus ("set top boxes"), produse electronice, programabile, de larg consum, PC-uri de rețea, minicalculatoare, computere mainframe, medii de calcul distribuite, care includ oricare dintre sistemele sau dispozitivele de mai sus, precum şi altele asemănătoare.

Invenția poate fi descrisă în contextul general al instrucțiunilor executabile pe computer, cum ar fi module de program ce sunt executate de un calculator. În general, modulele de program includ rutine, programe, obiecte, componente, structuri de date, şi aşa mai departe, care execută anumite seturi de instrucțiuni ("task"-uri) sau care implementează anumite tipuri abstracte de date. Invenția poate fi pusă în practică, de asemenea, în medii de calcul distribuite, unde lucrările sunt executate prin dispozitive de procesare de la distanță, care sunt legate printr-o rețea de comunicații. Într-un mediu de calcul distribuit, modulele de program pot fi localizate pe ambele medii de stocare pe computer, locale şi aflate la distanță, ce includ dispozitivele de stocare în memorie.

În legătură cu fig. 1, un sistem tipic pentru implementarea invenției include un dispozitiv de calcul cu destinație generală, sub forma unui computer **110**. Componentele calculatorului **110** pot include, dar nu sunt limitate de acestea, o unitate de procesare **120**, o memorie de sistem **130**, precum şi o magistrală de sistem **121**, care cuplează diverse componente ale sistemului ce include memoria de sistem, la unitatea de procesare **120**. Magistrala de sistem **121** poate fi oricare dintre diversele tipuri de structuri de magistrale ce includ o magistrală de memorie sau un controler de memorie, o magistrală de periferice, precum şi o magistrală locală ce utilizează oricare din varietatea de arhitecturi de magistrale. Cu titlu de exemplu, iar nu ca limitare, astfel de arhitecturi includ magistrala ISA (Industry Standard Architecture - Arhitectura standard industrială), magistrala MCA (Micro Channel Architecture - Arhitectura de micro canal), magistrala EISA (Enhanced ISA - ISA sporit), magistrala locală VESA (Video Electronics Standards Association - Asociația de standarde de electronică video), magistrala AGP (Accelerated Graphics Port - Port de grafică accelerată), precum şi magistrala PCI (Peripheral Component Interconnect - Interconectare de componente periferice), cunoscută de asemenea drept magistrala "Mezzanine".

Computerul **110** include, într-un mod caracteristic, o varietate de medii citibile cu ajutorul calculatorului. Mediile citibile cu ajutorul calculatorului pot fi orice medii disponibile, care pot fi accesate de către calculatorul **110**, şi includ atât medii volatile şi medii nonvolatile, cât şi medii movibile şi amovibile. Cu titlu de exemplu şi nu de limitare, mediile citibile cu ajutorul

calculatorului pot conține medii de stocare pe calculator şi medii de comunicație. Mediile de stocare pe calculator includ atât medii volatile şi nonvolatile, cât şi medii movibile şi amovibile, implementate în oricare metodă sau tehnologie pentru stocarea informațiilor, cum ar fi instrucțiuni citibile cu ajutorul calculatorului, structuri de date, module de program sau alte date. Mediile de stocare pe calculator includ, dar nu sunt limitate de acestea, RAM, ROM, EEPROM, memorie "flash" sau alte tehnologii pentru memorii, CD-ROM, DVD-uri (Digital Versatile Disk - disc digital versatil) sau alte stocări optice pe disc, casete magnetice, bandă magnetică, stocare magnetică pe disc sau alte dispozitive magnetice de stocare, sau orice alt mediu care poate fi utilizat pentru a stoca informația dorită şi care pot fi accesate de către computerul **110**. Mediile de comunicație cuprind, înt-un mod caracteristic, instrucțiuni citibile cu ajutorul calculatorului, structuri de date, module de program sau alte date dintr-un semnal de date modulat, cum ar fi o undă purtătoare sau alt mecanism de transport şi include orice medii de livrare a informației. Termenul "semnal de date modulat" înseamnă un semnal care are una sau mai multe din caracteristicile acestuia, setate sau modificate într-o astfel de manieră, încât să codifice informația din semnal. Cu titlu de exemplu şi nu de limitare, mediile de comunicație includ medii cablate, cum ar fi o rețea cablată sau o conexiune cablată direct, precum şi medii "wireless" (fără fir), cum ar fi medii acustice, RF, în infraroşu şi alte medii wireless. Combinațiile oricărora de mai sus trebuie să fie incluse, de asemenea, în sfera mediilor citibile cu ajutorul calculatorului.

Memoria de sistem **130** include mediile de stocare pe calculator, sub forma memoriei volatile şi/sau a memoriei nonvolatile, cum ar fi ROM-ul **131** (Read Only Memory - memorie numai de citire) şi RAM-ul **132** (Random Access Memory - memorie cu acces aleatoriu). Un BIOS (Basic Input/Output System - sistem de bază de intrare/ieşire) **133**, conținând rutine de bază care ajută la transferarea informației între elementele din interiorul calculatorului **110**, cum ar fi cele pe durata "start-up"-ului, este într-un mod caracteristic stocat în ROM-ul **131**. RAM-ul **132** conține, în mod caracteristic, date şi/sau module de program care sunt accesibile imediat şi/sau în mod necesar, fiind funcționale la unitatea de procesare **120**. Cu titlu de exemplu şi nu de limitare, fig. 1 ilustrează sistemul de operare **134**, programele de aplicație **135**, alte module de program **136** şi date de program **137**.

Calculatorul **110** poate să includă, de asemenea, alte medii movibile/amovibile, volatile/nonvolatile de stocare în calculator. Numai cu titlu de exemplu, fig. 1 ilustrează o unitate de hard disk **141**, care citeşte sau scrie pe medii magnetice amovibile, nonvolatile, o unitate de disc magnetic **151**, care citeşte sau scrie pe un disc magnetic **152** movibil, nonvolatil, precum şi o unitate de disc optic **155**, care citeşte sau scrie pe un disc optic **156**, movibil, nonvolatil, cum ar fi un CD-ROM sau alte medii optice. Alte medii movibile/amovibile, volatile/nonvolatile de stocare în calculator, care pot fi utilizate în mediul tipic de operare, includ, dar nu sunt limitate la acestea, casete de bandă magnetică, carduri de memorie flash, discuri digitale versatile, bandă video digitală, RAM cu semiconductoare, ROM cu semiconductoare, precum şi altele asemănătoare. Unitatea de hard disc **141** este conectată, în mod caracteristic, la magistrala de sistem **121**, printr-o interfață de memorie amovibilă, cum ar fi interfața **140**, iar unitatea de disc magnetic **151** şi unitatea de disc optic **155** sunt conectate, în mod tipic, la magistrala de sistem **121**, printr-o interfață de memorie movibilă, cum ar fi interfața **150**.

Unitățile şi mediile asociate, de stocare în computer, discutate mai sus şi ilustrate în fig. 1, asigură stocarea instrucțiunilor citibile cu ajutorul calculatorului, a structurilor de date, a modulelor de program, precum şi a altor date pentru calculatorul **110**. În fig. 1, de exemplu, unitatea de hard disc **141** este ilustrată ca stocând sistemul de operare **144**, programele de aplicație **145**, alte module de program **146**, precum şi datele de program **147**. De remarcat că

aceste componente pot fie să fie la fel, fie să fie diferite de sistemul de operare **134**, de programele de aplicaţie **135**, de alte module de program **136** şi de datele de program **137**. Sistemul de operare **144**, programele de aplicaţie **145**, alte module de program **146**, precum şi datele de program **147**, au indicate diferite numere aici, pentru a ilustra că, la un minimum, acestea sunt copii diferite. Un utilizator poate să introducă comenzi şi informaţii în calculatorul **110**, prin dispozitive de intrare, cum ar fi o tabletă (digitizor electronic) **164**, un microfon **163**, o tastatură **162** şi un dispozitiv de pointare **161**, în mod obişnuit referit ca fiind "mouse", "trackball" sau "touch pad". Alte dispozitive de intrare (nu sunt arătate) pot include un "joystick", "game pad", reflector parabolic asimetric pentru satelit, scaner sau altele asemănătoare. Acestea, precum şi alte dispozitive de intrare, sunt deseori conectate la unitatea de procesare **120**, printr-o interfaţă de intrare utilizator **160**, care este cuplată la magistrala de sistem, dar care poate fi conectată de către altă interfaţă sau alte structuri de magistrale, cum ar fi un port paralel, un port pentru jocuri sau o magistrală serială universală (USB). Un monitor **191** sau alt tip de dispozitive de afişare este, de asemenea, conectat la magistrala de sistem **121**, printr-o interfaţă, cum ar fi o interfaţă video **190**. Monitorul **191** poate, de asemenea, să fie integrat, împreună cu un panou "touch-screen" **193** sau ceva asemănător, care poate să introducă o intrare digitizată, cum ar fi scrisul de mână, în sistemul calculatorului **110**, printr-o interfaţă, cum ar fi interfaţa de "touch-screen" **192**. De notat faptul că monitorul şi/sau panoul de touch-screen pot fi, din punct de vedere fizic, cuplate la o carcasă în care este încorporat dispozitivul de calcul **110**, cum ar fi într-un calculator personal de tip tabletă, la care panoul de touch screen **193** serveşte, în fond, drept tableta **164**. În afară de aceasta, computere cum ar fi dispozitivul de calculat **110**, pot să includă, de asemenea, alte dispozitive periferice de ieşire, cum ar fi difuzoarele **195** şi imprimanta **196**, care pot fi conectate, printr-o interfaţă **194**, de periferice de ieşire sau altele asemănătoare.

Calculatorul **110** poate funcţiona într-un mediu de reţea, ce utilizează conecţiuni logice, la unul sau mai multe computere aflate la distanţă, cum ar fi calculatorul **180**, aflat la distanţă. Calculatorul **180**, aflat la distanţă, poate fi un calculator personal, un server, un "router", un PC de reţea, un dispozitiv "peer" sau alte noduri comune de reţea, şi include, în mod caracteristic, multe sau toate din elementele descrise mai sus, referitoare la computerul **110**, cu toate că, numai un dispozitiv **181** de stocare în memorie a fost ilustrat în fig. 1. Conexiunile logice, reprezentate în fig. 1, includ un LAN (Local Area Network - reţea de zonă locală) **171** şi un WAN (Wide Area Network - reţea de zonă largă) **173**, dar pot, de asemenea, să includă alte reţele. Astfel de medii de conectare în reţea sunt comune în birouri, în reţele de calculatoare pe arie largă din întreprinderi, în intraneturi şi pe Internet.

Atunci când se utilizează într-un mediu de conectare în reţea LAN, computerul **110** este conectat la LAN-ul **171**, printr-o interfaţă de reţea sau prin adaptorul **170**. Atunci când se utilizează într-un mediu de conectare în reţea WAN, computerul **110** include, în mod caracteristic, un modem **172** sau alte mijloace pentru stabilirea comunicaţiilor prin WAN-ul **173**, cum ar fi Internetul. Modemul **172**, care poate fi intern sau extern, poate fi conectat, la magistrala de sistem **121**, prin interfaţa de intrare utilizator **160** sau printr-un alt mecanism adecvat. Într-un mediu de conectare în reţea, modulele de program descrise, referitore la calculatorul **110** sau porţiuni din acestea, pot fi stocate într-un dispozitiv de stocare în memorie, aflat la distanţă. Cu titlu de exemplu şi nu de limitare, fig. 1 ilustrează programe **185** de aplicaţie aflate la distanţă, ca rezidente pe dispozitivul de memorie **181**. Va fi de apreciat faptul că respectivele conexiuni de reţea arătate sunt tipice, şi pot fi utilizate şi alte mijloace de stabilire a legăturii de comunicaţii dintre calculatoare.

Un aspect al prezentei invenții este, în general, îndreptat spre a permite codului de program, cum ar fi o aplicație sau o componentă a sistemului de operare, să comunice instrucțiunile de desenare, precum şi alte informații (de exemplu, bitmap-urile de imagine) la componentele de grafică, pentru a livra ieşirea grafică pe afişajul sistemului. În acest scop, invenția de față prevede un limbaj de marcare, împreună cu un set de elemente de conformație, precum şi alte elemente, prevede un sistem de grupare şi de compoziție, precum şi integrarea cu un sistem de proprietăți generale dintr-un model obiect, pentru a permite programelor să populeze un graf scenic cu structuri de date, primitive de desenare (comenzi), precum şi alte date referitoare la grafică. Atunci când este procesat, graful scenic are ca rezultat grafică afişată pe ecran.

Fig. 2 reprezintă o arhitectură generală, stratificată **200**, în care poate fi implementată prezenta invenție. Aşa după cum este reprezentat în fig. 2, codul de program **202** (de exemplu, un program de aplicație sau o componentă a sistemului de operare sau altele asemănătoare) poate fi dezvoltat să producă date de grafică pe una sau mai multe căi diverse, inclusiv prin formarea de imagine **204**, prin elementele de grafică vectorială **206**, şi/sau prin apeluri de funcții/metode plasate direct la un strat **212**, al interfeței de programare a aplicației vizualului (API). Interacțiunea directă cu stratul API este descrisă mai departe, în cererea de brevet de invenție dependentă, menționată mai sus, intitulată "Interfețe grafice vizuale şi scenice".

În general, formarea de imagine **204** asigură, codul de program **202**, cu un mecanism pentru încărcarea, editarea şi salvarea de imagini, cum ar fi bitmap-urile. Aceste imagini pot fi utilizate de către alte părți ale sistemului şi există, de asemenea, o cale de a utiliza codul de desenare al primitivelor, pentru a desena direct pe o imagine.

În conformitate cu un aspect al prezentei invenții, elementele de grafică vectorială **206** asigură o altă cale de a desena grafică, compatibilă cu restul din modelul obiect (aşa cum este descris mai jos). Elementele **206** de grafică vectorială pot fi create printr-un limbaj de marcare, care prelucrează un sistem **208**, de elemente/proprietăți, şi un sistem prezentator **210** să facă apeluri specifice la stratul API al vizualului, **212**. Aşa după cum este descris mai jos, în legătură cu fig. 26, în general, elementele **206** de grafică vectorială sunt analizate, gramatical, în obiecte ale modelului obiect din care este desenat un graf scenic, care pot fi asigurate la graful de decor printr-un nivel de elemente, prin sistemul **208** de elemente/proprietăți şi prin sistemul prezentator **210**, sau poate fi asigurat într-o manieră mai eficientă la un nivel de resurse, aşa cum este, de asemenea, descris mai jos.

Într-o implementare, arhitectura **200**, a stratului de grafică, include un motor **214**, de compoziție şi de animație de nivel ridicat, care include sau este asociat, pe de altă parte, cu o structură **216**, de date de memorare ascunsă. Structura **216** de date de memorare ascunsă, conține un graf scenic, ce cuprinde obiecte aranjate în mod ierarhic, care sunt administrate în conformitate cu un model obiect definit, aşa după cum este descris mai jos. În general,stratul **212** API al vizualelor asigură codul de program **202** (precum şi sistemul prezentator **210**) cu o interfață pentru structura **216**, de date de memorare ascunsă, incluzând abilitatea de a crea obiecte, de a deschide şi de a închide obiecte, pentru a le furniza date, şi aşa mai departe. Cu alte cuvinte, motorul **214** de compoziție şi de animație de nivel superior etalează un strat API de medii unificate, **212**, prin care dezvoltatorii pot să-şi exprime intențiile referitoare la grafică şi la medii, pentru a afişa informații de grafică, şi asigură o platformă fundamentală cu destule informații, astfel că platforma poate optimiza utilizarea hardware-lui pentru codul de program. De exemplu, platforma fundamentală va fi răspunzătoare de memorarea ascunsă, de negocierea de resurse şi de integrarea mediilor.

Într-o implementare, motorul **214** de compoziţie şi de animaţie de nivel superior trece un şir de instrucţiuni şi posibile alte date (de exemplu, pointer-e la bitmap-uri), la un motor **218** de compoziţie şi de animaţie rapid, de nivel inferior. Aşa după cum sunt utilizaţi în cele de faţă, termenii de "nivel superior" şi de "nivel inferior" sunt similari acelora utilizaţi în alte scenarii de calcul, în care, în general, cu cât este mai scăzută componenta de software faţă de componentele de nivel superior, cu atât este mai apropiată acea componentă de elementele hardware. Astfel, de exemplu, informaţia de grafică, trimisă de la motorul **214** de compoziţie şi de animaţie de nivel superior, poate fi recepţionată la motorul **218**, de compoziţie şi de animaţie de nivel inferior, unde informaţia este utilizată pentru a trimite datele de grafică la subsistemul de grafică, ce include hardware-ul **222**.

Motorul **214** de compoziţie şi de animaţie de nivel superior, împreună cu codul de program **202**, construieşte un graf de decor, pentru a reprezenta un decor de grafică, asigurat de codul de program **202**. De exemplu, fiecare element ce este desenat, poate fi încărcat, împreună cu instrucţiunile de desenare, pe care sistemul îl poate ascunde în structura de date **216**, a grafului scenic. Aşa după cum va fi descris mai jos, există un număr de căi diverse, pentru a specifica această structură de date **216**, precum şi ceea ce este desenat. În plus, motorul **214** de compoziţie şi de animaţie de nivel superior integrează, sistemele **220**, cu temporizarea şi animaţia, pentru a asigura controlul declarativ (sau un altul) al animaţiei (de exemplu, intervalele de animaţie), precum şi controlul temporizării. De remarcat că sistemul de animaţie permite valori animate, pentru a fi trecute, în esenţă, oriunde în sistem, fiind incluse, de exemplu, la nivelul de proprietăţi ale elementelor, **208**, din interiorul stratului **212**, API, al vizualelor, precum şi la oricare alte resurse. Sistemul de temporizare este expus la nivelurile de elemente şi de vizuale.

Motorul **218** de compoziţie şi de animaţie de nivel inferior administrează compunerea, animarea şi remiterea scenei, care este apoi prevăzută la subsistemul **222**, de grafică. Motorul **218** de nivel inferior alcătuieşte remiterile pentru scenele aplicaţiilor multiple, iar împreună cu componentele de remitere, implementează remiterea reală, a graficii, pe ecran. De remarcat totuşi că, din când în când, poate fi necesar şi/sau avantajos, ca unele dintre remiteri să se facă la niveluri mai ridicate. De exemplu, în timp ce service-ul straturilor inferioare necesită aplicaţii multiple, straturile superioare sunt instanţiate imediat pe o bază de per-aplicaţie, prin care este posibil, prin mecanismele **204**, de formare de imagini ("imaging"), să se efectueze remiteri consumatoare de timp sau remiteri specifice aplicaţiei, la niveluri superioare, şi să treacă referirile la un bitmap, la nivelurile inferioare.

Aşa cum este descris mai jos, modelul de remitere este împărţit între elementele **206**, de grafică vectorială de înalt nivel, bazate pe comenzi, şi obiectele de nivel scăzut, create de către stratul API al vizualului, **212**, utilizat în structura **216**, de date, a grafului scenic. Acest lucru asigură o valoare însemnată a corelaţiei dintre elementele de nivel ridicat, ale prezentei invenţii, şi obiectele de nivel scăzut. Ceea ce urmează, descrie o implementare a modelului obiect al grafului scenic.

Fig. 3 şi 4 prezintă grafuri scenice model, **300**, şi, respectiv, **400**, ce includ un obiect de bază menţionat ca fiind un vizual. În general, un vizual conţine un obiect care reprezintă o suprafaţă virtuală la utilizator şi are o reprezentare vizuală pe display. Aşa după cum este reprezentat în fig. 5, un vizual de clasă de bază asigură funcţionalitatea de bază, pentru alte tipuri de vizuale, adică clasa **500**, a vizualului, este o clasă de bază abstractă, din care derivă tipurile vizualului (de exemplu, **501**-**506**).

Aşa după cum este reprezentat în fig. 3, un vizual **302**, de nivel de vârf (sau rădăcină), este conectat la un obiect **304**, manager de vizuale, care are, de asemenea, o relaţie (de exemplu, printr-un pretext convenabil) cu o fereastră (HWnd) **306** sau cu o unitate similară, la care datele graficului sunt extrase pentru codul de program. Managerul de vizuale ("VisualManager") **304** administrează desenul vizualului de nivel de vârf (precum şi "copiii" acestui vizual) la această fereastră **306**. Fig. 6 prezintă "VisualManager"-ul ca unul dintr-un set de alte obiecte **620**, din modelul obiect al sistemului de grafică descris în cele de faţă.

Pentru a desena, managerul de vizuale, **304**, procesează (de exemplu, parcurge sau transmite) graful scenic, aşa după cum este planificat de către un dispecer **308**, şi prevede instrucţiuni de grafică, precum şi alte date, la componenta **218**, de nivel scăzut (fig. 2), pentru fereastra lui corespunzătoare, **306**, după cum este descris, la modul general, în cererile de brevet **US 2003/0076328**, **US 2003/0132937** şi **US 2003/0076329**. Procesarea grafului scenic va fi planificată în mod obişnuit, de către dispecerul **308**, la o viteză care este relativ mai scăzută decât viteza de reîmprospătare a componentei **218**, de nivel inferior şi/sau a subsistemului **222**, de grafică. Fig. 3 arată un număr de vizuale "copii" **310-315**, aranjate în mod ierarhic sub vizualul **302**, de nivel de vârf (rădăcină), din care unele sunt reprezentate ca fiind populate prin contextele de desenare **316** şi **317** (prezentate sub formă de căsuţe cu linii întrerupte, pentru a reprezenta natura lor temporară) cu liste de instrucţiuni asociate, **318**, şi, respectiv, **319**, ce conţin, de exemplu, primitive de desenare, precum şi alte vizuale. Vizualele pot, de asemenea, să conţină alte informaţii de proprietăţi, aşa cum sunt arătate la următoarea clasă exemplu de vizuale:

```
public abstract class Visual : VisualComponent
 {
    public Transform Transform { get; set; }
    public float Opacity { get; set; }
    public BlendMode BlendMode { get; set; }
    public Geometry Clip { get; set; }
    public bool Show { get; set; }
   public HitTestResult HitTest (Point point);
   public bool IsDescendant (Visual visual);
   public static Point TransformToDescendant(
          Visual reference,
          Visual descendant,
          Point point);
   public static Point TransformFromDescendant(
          Visual reference,
          Visual descendant,
           Point point);
    public Rect CalculateBounds ( ); / / Loose bounds
    public Rect CalculateTightBounds ( ); / /
    public bool HitTestable { get; set; }
    public bool HitTestIgnoreChildren { get; set; }
    public bool HitTestFinal { get; set; }
 }
```

O transformare setată de către proprietatea transformatei defineşte sistemul de coordonate pentru subgraful unui vizual. Sistemul de coordonate, dinaintea transformării, este denumit sistem de coordonate de pretransformare, cel de după transformare este denumit sistem de coordonate de post-transformare, adică un vizual cu o transformare este echivalent cu un vizual cu un nod de transformare, drept unul "părinte". Fig. 7 prevede, în general, un exemplu de transformare, ce identifică sistemele de coordonate de pretransformare şi de post-transformare, privitoare la un vizual. Pentru a obține sau a seta transformarea unui vizual, poate fi folosită proprietatea "Transform " (transformă).

De remarcat că transformatele de coordonate pot fi aplicate, pe o cale uniformă, la orice, chiar dacă ar fi într-un bitmap. De notat că aceasta nu înseamnă că transformările se folosesc întotdeauna la bitmap-uri, dar ceea ce se remite este afectat în mod egal de către transformate. În chip de exemplu, dacă utilizatorul desenează un cerc cu trăgător rotund, care este de lățime de un "inch" (țol) şi apoi se aplică o scară gradată pe direcția X din cele două direcții, la acest cerc, trăgătorul va fi de lățime doi inch la stânga şi la dreapta, şi de numai un inch lățime în sus şi în jos. Acest lucru este referit uneori ca fiind o transformată de compoziție sau de bitmap (ca opusă unui schelet sau unei scale de geometrie care afectează numai geometria). Fig. 8A este o reprezentare a transformării de scalare, cu o imagine netransformată **800**, ce apare în stânga, precum şi o imagine transformată **802**, cu o scală neuniformă, ce apare în dreapta. Fig. 8B este o reprezentare a transformării de scalare, cu imaginea netransformată **800**, ce apare în stânga, precum şi o imagine transformată **804**, cu o scalare geometrică ce apare în dreapta.

Ținând seama de transformarea de coordonate a unui vizual, "TransformToDescendant" (transformă la descendent) transformă un punct de la vizualul de referință la un vizual descendent. Punctul este transformat din spațiul de coordonate de post-transformare a vizualului de referință, în spațiul de coordonate de post-transformare a vizualului descendent. "TransformFromDescendant" (transformă din descendent) transformă un punct din vizualul descendent, începând cu lanțul de părinți, la vizualul de referință. Punctul este transformat din spațiul de coordonate de post-transformare a vizualului descendent, în spațiul de coordonate de post-transformare a vizualului de referință. Metoda "CalculateBounds" (calculează bordurile) returnează căsuța de delimitare a conținutului "Visual"-ului din spațiul de coordonate de post-transformare. De remarcat că poate exista o versiune alternativă a API-ului, unde sunt permise mai multe specificații specifice, despre felul cum transformata de la un vizual este interpretată pe durata unei transformări de coordonate. De exemplu, transformata din vizualul de referință şi descendent poate sau nu poate fi luată în considerație. În această alternativă, există astfel patru alternative, de exemplu, coordonatele pot fi transformate din spațiu de pretransformare în spațiu de pretransformare, din spațiu de pretransformare în spațiu de post-transformare, din spațiu de post-transformare în spațiu de pretransformare, şi din spațiu de post-transformare în spațiu de post-transformare. Acelaşi concept se aplică la testarea de atingere a țintei, de exemplu, testarea de atingere a țintei poate fi startată la spațiul de coordonate al transformatelor de pretransformare sau de post-transformare, iar rezultatele testului de atingere a țintei ar putea fi în spațiul de coordonate de pretransformare sau în cel de post-transformare.

Proprietatea de a decupa setează (şi obține) regiunea de decupare a unui vizual. Orice "Geometry" (geometrie) (clasa de geometrie este descrisă mai jos, în legătură cu fig. 12) poate fi utilizată ca o regiune de decupare, iar regiunea de decupare este aplicată în spațiul de coordonate de post-transformare. Într-o implementare, o setare de referință pentru regiunea de decupare este nulă, adică fără decupare, care poate fi gândită ca fiind un dreptunghi infinit de mare de decupare, de la $(-\infty, -\infty)$ la $(+\infty, +\infty)$.

Proprietatea de "Opacity" (opacitate) obține/setează valoarea opacității unui vizual, astfel că respectivul conținut al vizualului este îmbinat pe suprafața de desen, pe baza valorii opacității şi a modului de îmbinare selectat. Proprietatea de "BlendMode" (mod de îmbinare) poate fi utilizată pentru a seta (sau a obține) modul de îmbinare care este folosit. De exemplu, o valoare ("alpha") a opacității poate fi setată între 0,0 şi 1,0, cu setul alpha liniar de îmbinare, drept mod de setare, de exemplu, " Color = alpha * foreground color + (1.0-alpha) * background color) ". Alte servicii, cum ar fi proprietăți de efecte speciale, pot fi incluse într-un vizual, de exemplu, pată, monocrom şi aşa mai departe.

Diversele servicii (incluzând transformata, opacitatea, decuparea) pot fi introduse şi scoase dintr-un context de desenare, iar operațiile de introducere/scoatere ("push/pop") pot fi incluse una într-alta, atâta timp cât un apel de scoatere ("pop") se potriveşte cu un apel de introducere ("push"). De exemplu, "PushTransform(...); PushOpacity(...); PopTransform(...);" este ilegal, deoarece înaintea apelului de "PopTransform", trebuie să fie chemat "PopOpacity".

Metoda de "PushTransform" introduce o transformare. Operațiile de desenare, care urmează, sunt executate, ținând seama de transformarea introdusă. "PopTransform" scoate transformarea introdusă de către apelul potrivit "PushTransform":

void PushTransform (Transform transform);
void PushTransform (Matrix matrix);
void PopTransform ();.

În mod similar, metoda " PushOpacity " introduce o valoare a opacității. Operațiile de desenare, care urmează, sunt remise pe o suprafață temporară, împreună cu valoarea specificată a opacității şi apoi sunt combinate în decor. " PopOpacity" extrage opacitatea introdusă de către apelul potrivit de "PushOpacity"

void PushOpacity (float opacity);
void PushOpacity (NumberAnimationBase opacity);
void PopOpacity ();.

Metoda "PushClip" introduce o geometrie de decupare. Operațiile de desenare, care urmează, sunt decupate la geometrie. Decuparea este aplicată la spațiul de post transformare."PopClip" extrage regiunea de decupare introdusă de către apelul potrivit de "PushClip":

void PushClip (Geometry clip);
void PopClip ();.

De notat faptul că operațiile de introducere pot fi incluse în mod arbitrar una în alta, atâta timp cât operațiile de scoatere sunt combinate cu o introducere. De exemplu, este valabil următorul lucru:

```
PushTransform(...);
DrawLine(...);
PushClip (...);
DrawLine(...);
PopClip ( );
PushTransform (...);
DrawRect(...);
PopTransform( );
PopTransform( );
```

Testarea de atingere a țintei este efectuată în spațiul de coordonate de post-transformare, iar returnarea unei identități a fiecărui vizual testabil la atingerea țintei, adică este atinsă ținta, de exemplu, atunci când este detectat un trăgător sau un "click" de "mouse". O versiune alternativă a interfeței poate permite, pentru testarea de atingere a țintei, startarea la spațiul de coordonate de post-transformare, privitoare la vizual, unde este startat testul de atingere a țintei. Vizualele care sunt țintite sunt returnate în ordinea primul întâlnit, de la dreapta la stânga. Testarea de atingere a țintei poate fi controlată cu diverse indicatoare (flag) ce includ "HitTestable", care determină dacă vizualul este testabil din punct de vedere al atingerii țintei (referința este adevărată), şi " HitTestFinal"-ul, care determină dacă testarea de atingere a țintei se opreşte atunci când vizualul este țintit, adică dacă un "Visual" este țintit, iar proprietatea de "HitTestFinal" a vizualului este adevărată, testarea la atingerea țintei se întrerupe şi se returnează rezultatele colectate la acest punct (referința este falsă). Un alt indicator este "HitTestIgnoreChildren", care determină dacă "copiii" unui vizual ar trebui luați în considerare, atunci când testarea de atingere a țintei este efectuată pe un vizual (referința este falsă).

Un "ProxyVisual" este un vizual care poate fi adăugat mai mult decât o dată în graful scenic. Deoarece orice vizual care este referit de către un "ProxyVisual" poate fi atins prin căi multiple de la rădăcină, serviciile de citire ("TransformToDescendent", "TransformFromDescendent" şi "HitTest") nu lucrează printr-un "ProxyVisual". În esență, există o cale canonică de la orice vizual la rădăcina arborelui de vizuale, iar această cale nu include niciun "ProxyVisual".

Aşa cum este reprezentat în fig. 5, diversele tipuri de vizuale sunt definite în modul obiect, ce includ vizualele de recipient ("ContainerVisual") **501**, vizualele de desenare ("DrawingVisual") **502**, vizualele de validare ("ValidationVisual") **503**, vizualele de suprafață ("SurfaceVisual") **504** şi vizualele Hwnd ("HwndVisual") **505**. Tabelul de mai jos descrie metodele model ale unui vizual de desenare:

```
public class DrawingVisual: Visual
{
    public DrawingVisual ( );
    public IDrawingContext Open ( );
    public IDrawingContext Append ( );
}
```

Un "DrawingVisual" este un recipient pentru conținut grafic (de exemplu, linii, text, imagini şi aşa mai departe). De remarcat faptul că este posibil să se adauge un "Visual" într-un "DrawingVisual", dar la unele implementări, acest lucru nu este permis. "DrawingVisual"-ul **502** include o metodă de deschidere ("Open"), care returnează un "IdrawingContext" care poate fi utilizat pentru a popula "DrawingVisual"-ul, de exemplu, cu alte vizuale şi cu primitive de desenare, aşa cum este descris mai jos. La o implementare, pentru diverse motive, descrise, de asemenea, mai jos, un "DrawingVisual" poate fi deschis numai o dată, pentru a popula contextul lui de desenare; cu alte cuvinte, un astfel de "DrawingVisual" este invariabil.

După ce a fost populat "DrawingVisual"-ul, acesta este închis, utilizând o metodă "Close" (închide), de exemplu pe contextul de desenare. De notat faptul că un apel "Open" (deschide) poate şterge orice conținut ("copiii") al unui vizual, totuşi la o implementare alternativă, este prevăzută o metodă "Append " (anexează), pentru a deschide un vizual curent, într-o manieră care se anexează la acest vizual. Cu alte cuvinte, un apel de "OpenForAppend" (deschide pentru anexare) lucrează la fel ca "Open", cu excepția faptului că, respectiv, conținutul curent al lui "DrawingVisual" nu este şters la deschidere.

Ceea ce urmează este un exemplu al felului cum un context de desenare este utilizat pentru a popula un vizual:

```
ContainerVisual cv1 = new ContainerVisual ( );
DrawingVisual dv1 = new DrawingVisual ( );
// Deschide un context de desenare
 Contextul // va fi închis în mod automat atunci când
// se iese din blocul de utilizare.
Acesta va înlocui // de asemenea orice conținut care ar putea deja
// să fie în dv1.
using (IDrawingContext dc = dv1. Open ( ) )
{
    dc. DrawLine (new Pen (Brushes. Blue), new Point(…), new Point(…));
}

// Adaugă dv1 la colecția "copil" din cv1
cv1. Children. Add (dv1);

// Adaugă un alt vizual arbitrar la cv1
cv1. Children. Add (someOtherVisual);

// Crează un alt DrawingVisual
DrawingVisual dv2 = new DrawingVisual ( );

using (IDrawingContext dc = dv2. Open ( ) )
{
    // Aceasta setează un nou sistem de coordonate
    // unde orice lucru este de două ori mai mare
    dv. PushTransform (new Scale (2.0, 2.0) );

    // Această linie este desenată în noul scalat
    // sistem de coordonate.
    dc. DrawLine (new Pen (Brushes.Red), new Point (…), new Point (…));

    // Aceasta revine la sistemul de coordonate inițial.
    dv. PopTransform ( );

    dc.DrawLine (new Pen (Brushes.Green), new Point (…), new Point (…) );
}

// Adaugă dv2 la colecția "copil" din cv1;
cv1. Children. Add (dv2);
```

În general, un "ValidationVisual" (vizual de validare) **503** este în mod conceptual similar cu un "DrawingVisual", cu excepția faptului că "ValidationVisual"-ul este populat atunci când sistemul cere ca acesta să fie completat, în locul faptului că, respectiv, codul de program cere să-l populeze. De exemplu, aşa după cum este descris în cererea de brevet de invenție **US 2003/0076329**, motorul **214**, de compoziție şi de animație de nivel ridicat (fig. 2), poate

invalida datele grafului scenic, deoarece sunt necesare resursele, cum ar fi atunci când o porțiune din graful de decor nu este vizibilă. De exemplu, dacă unele porțiuni sunt defilate în afara display-ului, sunt decupate, şi aşa mai departe. Dacă mai târziu, sunt necesare datele invalidate ale grafului scenic, codul de program **202**, apelat, va fi rechemat pentru a redesena (a valida) porțiunea invalidată din graful scenic. În acest sens, un scenariu tipic de utilizare este cel pentru un cod de program, pentru a sub-clasa "ValidationVisual"-ul şi pentru a trece peste metoda de "OnValidate". Când sistemul apelează metoda de "OnValidate", este introdus un context de desenare, iar programul care utilizează contextul de desenare, repopulează "ValidationVisual"-ul.

Exemplul de mai jos arată o cale pentru a implementa un singur vizual de validare ("ValidationVisual"), de exemplu, unul care desenează o linie cu anumită culoare. Culoarea liniei poate fi modificată prin apelarea lui "SetColor" (setează culoarea). Pentru a forța actualizarea "ValidationVisual"-ului, "SetColor"-ul apelează pe "Invalidate" (invalidează), pentru a forța subsistemul de grafică să revalideze "ValidationVisual"-ul:

```
public class MyValidationVisual : ValidationVisual
{
    public override void Onvalidate (IDrawingContext dc )
    {
        dc. DrawLine (m_color, ...),;
    }

    public void SetColor ( Color newColor )
    {
        m_color = color;
        Invalidate ( ); / / Forțează o redesenare a unui "ValidationVisual"
                        / / pentru a reflecta schimbarea de culoare.
    }
    private Color m_color
}
```

Acest exemplu arată cum să se utilizeze "ValidationVisual"-ul:

```
MyValidationVisual myVV = new MyValidationVisual ( );

container. Children. Add (myVV);

myVV. SetColor (new Color (...) );
```

Fig. 4 prezintă un graf scenic **400**, ca un exemplu la care "ContainerVisual"-ele şi "DrawingVisual"-ele sunt specificate într-un graf scenic şi au asociate date sub formă de primitive de desenare, de exemplu, în conformitate cu contextele de desenare. "ContainerVisual"-ul este un recipient pentru "Visual"-e, iar "ContainerVisual"-ele pot fi incluse unul în altul. "Copiii" unui "ContainerVisual" pot fi manevrați cu un "VisualCollection" (colecție de vizuale), returnat dintr-o proprietate "Children" (copii) a "VisualContainer"-ului. Ordinea

"Visual"-elor din "VisualCollection" determină în ce ordine sunt remise "Visual"-ele, adică "Visual"-ele sunt remise de la cel mai jos index la cel mai ridicat index, din spate în față (ordinea de pictare). De exemplu, cu ajutorul parametrilor specifici, împreună cu trei vizuale de desenare reprezentând dreptunghiuri roşii, verzi şi albastre, aflați în mod ierarhic sub un vizual de recipient, următorul cod ar rezulta din desenarea celor trei dreptunghiuri (translatare spre dreapta şi în jos), un dreptunghi roşu în spate, un dreptunghi verde în mijloc şi un dreptunghi albastru în față:

```
VisualCollection vc =  m_cv. Children;
vc. Add (new  DrawingVisual  ( ) );
vc. Add (new  DrawingVisual  ( ) );
vc. Add (new  DrawingVisual  ( ) );

for  (int  i  =  0;  i <  vc. Count;  i++)
{
      DrawingVisual  v  =  (DrawingVisual) (vc[ i ]);
      if  (v  ! =  null)
      {
            v.Transform = Transform.CreateTranslation (i  *  20.0f,  i * 20f);
            IDrawingContext  dc  =  v.Open  ( );
            dc.DrawRectangle (
                  new Brush (colors [ i ] ),
                  null,
                  new Point2D (0,  0),
                  new Point2D (100.0f,  100.0f) );
             v. Close (dc);
      }
}
```

Aşa cum este reprezentat în fig. 5, un alt tip de obiect vizual este un "SurfaceVisual" (vizual de suprafață) **504**. În general, aşa după cum este reprezentat în fig. 3, un obiect **315** al "SurfaceVisual"-ului completează o suprafață din memorie ("bitmap") **322**, pe care codul de program **202** (fig. 2) îl poate accesa. Codul **202** de program al clientului poate alimenta propria ei memorie de suprafață sau acesta poate cere ca memoria să fie alocată de către obiectul de suprafață.

Codul de program **202** are opțiunea de a deschide un "SurfaceVisual" şi de a obține un context de desenare **323**, în care codul de program **202** poate să scrie datele **324** de pixeli sau ceva asemănător şi să pună în mod direct aceşti pixeli pe suprafață. Acest lucru este reprezentat în fig. 3, de către linia întreruptă dintre obiectul de suprafață **322**, contextul de desenare **323** (prezentat ca o căsuță cu linii întrerupte, pentru a reprezenta natura lui temporară) şi datele pixeli **324**.

Codul de program **202** are şi o opțiune de a crea un administrator **330**, al vizualului de suprafață şi de a asocia un subgraf **332**, al vizualului, cu "SurfaceVisual"-ul **315**. Această opțiune este reprezentată în fig. 3, prin linia întreruptă dintre obiectul de suprafață **322** şi managerul **330**, al vizualului de suprafață. De remarcat faptul că subgraful **332**, al vizualului, poate, de asemenea, să încadreze alte vizuale de suprafață, după cum se arată, de asemenea, în fig. 3. Managerul **330**, al vizualului de suprafață (prezentat, de asemenea, drept un tip de alte obiecte din setul **620**, al fig. 6) parcurge subgraful **332**, al vizualului, pentru a actualiza bitmap-ul **322** al lui "SurfaceVisual". În continuare, este de notat faptul că această parcurgere

este planificată de către dispecerul **308**, iar pentru eficiență, poate fi oprită pentru a controla cât de des este actualizat acest bitmap **322**. Managerul **330**, al vizualului de suprafață, nu trebuie să parcurgă subgraful **322**, al vizualului, de fiecare dată şi/sau la aceeaşi viteză cu care managerul **302**, al vizualului de nivel de vârf, parcurge restul din graful scenic.

Ținând seama de suprafețe, aşa după cum sunt descrise mai departe, în legătură cu fig. 9A - 9C, în general, modelul de grafică de față permite, în felul acesta, combinarea unui set de vizuale la o suprafață permite remiterea de mod-imediat a primitivelor de vectori şi de bitmap într-o suprafață, permite combinarea unei suprafețe pe "desktop" sau pe o altă suprafață, şi permite controlarea suprafeței, dintr-o listă de suprafețe, care este utilizată pentru a se combina sau pentru a desena în interiorul ei. O listă de suprafețe este definită ca o colecție de una sau mai multe suprafețe (adică cadre/memorii tampon) de memorie fizică (de sistem sau video), utilizate pentru a stoca combinații de vizuale sau de desene grafice, sau ambele. Una dintre suprafețele din lista de suprafețe poate fi setată ca un "buffer" (memorie tampon) curent din spate, unde este efectuată desenarea şi/sau combinarea, iar una dintre suprafețele din lista de suprafețe este setată ca un "buffer" primar curent sau ca buffer frontal, care este utilizat pentru a se combina cu o altă țintă de remitere.

Suprafețele pot fi utilizate pe mai multe căi. Cu titlu de exemplu, fig. 9A prezintă combinarea la o suprafață. În fig. 9A, un obiect **900**, al managerului de vizuale de suprafață, conectează o listă **902**, de suprafețe, drept o țintă de remitere pentru un arbore **904**, de vizuale. Pe durata fiecărui ciclu de combinare, vizualele sunt combinate pe suprafața listei de suprafețe, care serveşte în mod curent drept buffer activ din spate, pentru lista de suprafețe. Suprafața, ce este combinată la acestea, poate include o suprafață deținută de către client/motorul **214**, de combinare de nivel ridicat (fig. 2), pentru scenariile de combinare în timpul procesului, poate include o suprafață deținută de către motorul **218**, de combinare de nivel scăzut, pentru scenariile unde clientul nu are nevoie de biți, dar motorul **218**, de combinare de nivel scăzut, are nevoie de acestea, pentru a combina suprafața de pe o altă țintă de remitere sau de pe o suprafață de proces în cruce, pentru scenarii unde clientul are nevoie de acces la biții de suprafață, însă motorul **218**, de combinare de nivel scăzut, are nevoie şi de suprafața pentru altă unitate de combinare.

Combinarea este controlată de către un serviciu de temporizare, care este ataşat la "VisualManager" (managerul de vizuale). Un exemplu de serviciu de temporizare este un mod manual, care ar trebui utilizat ca în exemplul de mai jos:

```
// creează un serviciu manual de temporizare şi ataşează un manager de vizuale
TimingService  timingService = new ManualTimingService(visualManager);

// combină arborele de vizuale la bufferul curent din spate al suprafeței
visualManager. Render ( );

foreach  (Tick  tick  in  timingService)
{
    // avansează buffer-ul din spate la următorul cadru al suprafeței
    surfaceList. NextFrame ( );

    // avansează durata arborelui de vizuale
    timingService. Tick (tick);

    // combină arborele de vizuale la buffer-ul curent din spate al suprafeței
    visualManager. Render ( );
}
```

Altă cale de a utiliza o suprafață este cea cu remitere de mod-imediat la o suprafață, printr-un context. Ataşarea unei liste de suprafețe la un vizual (un vizual de suprafață) permite remiterea de mod-imediat la suprafața din lista de suprafețe, care serveşte în mod curent drept buffer activ din spate, pentru lista de suprafețe. Această remitere este efectuată prin obținerea unui context de desenare de la vizualul de suprafață şi prin executarea comenzilor de desenare pe acest context, aşa după cum este descris mai sus. De remarcat că obținerea unui context de desenare blochează suprafața, astfel că alte operații de combinare nu pot fi efectuate asupra lui. Fiecare comandă de desenare este executată imediat, iar vectorii şi alte suprafețe pot fi desenate (amestecate) pe suprafață. Totuşi, alte vizuale nu pot fi desenate pe suprafață, dar în schimb, pot fi combinate pe suprafață prin asocierea acesteia cu un manager de vizuale, aşa după cum a fost descris anterior (de exemplu, în fig. 9A).

```
// ataşează o listă de suprafețe la un vizual
SurfaceVisual surfaceVisual = new SurfaceVisual (surfaceList);

// validează remiterea de mod-imediat (şi blochează) la suprafața buffer-ului
din spate
BaseDrawingContext dc = surfaceVisual. Open ( );

// desenează o linie (imediat) la buffer-ul curent din spate al
suprafeței
dc. DrawLine (pen, startPoint, endPoint);

// deblochează suprafața - am făcut-o cu remiterea de mod-imediat
surfaceVisual. Close (dc);
```

O altă utilizare pentru suprafețe este atunci când se combină o suprafață cu o altă țintă de remitere. În acest sens, odată ce lista de suprafețe este ataşată la un vizual de suprafață, atunci suprafața poate fi ataşată în arborele de vizuale, ca un nod, iar suprafața din lista de suprafețe, care serveşte în mod curent drept buffer primar sau frontal, poate fi combinată la o altă suprafață sau la "desktop". Acest lucru este ilustrat în fig. 9B şi în exemplul de mai jos:

```
// ataşează o listă de suprafețe la un vizual
SurfaceVisual surfaceVisual = new SurfaceVisual (surfaceList);

// Adaugă "surfaceVisual"-ul la un arbore de vizuale, pentru combinarea cu o
// altă țintă de remitere
rootVisual. Add (surfaceVisual);
```

Compunerea directă la/de la o suprafață este reprezentată în fig. 9C, unde posibilitățile descrise mai sus sunt combinate astfel că, respectiv, combinarea la suprafața buffer-ului din spate, a unei liste de suprafețe şi combinarea de la suprafața buffer-ului din față, a unei liste de suprafețe (de exemplu, la desktop) se întâmplă în mod simultan. De remarcat faptul că, pentru a elimina efectul video nedorit, cunoscut ca "tearing" (rupere a imaginilor), lista de suprafețe trebuie să aibă cel puțin două suprafețe, o suprafață a buffer-ului din față şi una a buffer-ului din spate. O suprafață utilizată ca în fig. 9C este deținută după cum se pare de către motorul **218**, de nivel scăzut, sau este o suprafață de procese în cruce, pentru a face ca, respectiv, combinarea din motorul **218**, de nivel scăzut, să se execute mai bine.

Suprafețele sunt construite ca obiecte independente, aşa după cum se arată din exemplele de mai jos ale constructorilor:

```
public  class  Surface
{
      / /  creează şi alocă o suprafață goală fără date inițiale
      public  Surface (int  width,
            int  height,
            int  dpi,
            PixelFormat  pixelFormat,
            SurfaceFlags  flags)

      / /  creează o suprafață, utilizând memoria furnizată
      public  Surface (int  width,
            int  height,
            int  dpi,
            PixelFormat  pixelFormat,
            IntPtr  pixels,  / /  memorie administrată pentru suprafață
            Int  stride)

      / /  creează dintr-o sursă  (adică  "Clone" – clonă)
      public  Surface (Surface  sourceSurface,
            SurfaceFlags  flags)

      / /  Creează ("Create") din "File" (fişier) sau din URL
      public  Surface (String  filename,
            SurfaceFlags  flags)

      / /  Creează din "Stream" (lanț)
      public  Surface (System. IO. Stream stream,
            SurfaceFlags  flags)

       / /  Creează din HBITMAP (care nu poate fi selectat într-un HDC)
      public  Surface (HBITMAP  hbitmap,  HPALETTE  hPalette)

       / /  Creează din HICON
       public  Surface (HICON  hicon)

       / /  proprietăți de tipul "read-only" (numai citire)
       public  Int  Width  { get;  }

       public  Int  Height  { get;  }

       public  Int  Dpi  { get;  }

       public  PixelFormat  Format  { get;  }

       public  int  Stride  { get;  }

       public  IntPtr  Buffer  { get;  }
}
```

```
public class SurfaceList
{
    // Creează o listă de suprafețe goale ( fără date inițiale ).
    public SurfaceList (int width,
        int height,
        int dpi,
        PixelFormat pixelFormat,
        int numSurfaces,
        SurfaceFlags flags)

    // Creează o "SurfaceList" (listă de suprafețe) care utilizează suprafețele
    // specificate
    // Toate suprafețele trebuie să aibă proprietăți identice (w, h,
    // dpi, etc.).
    public SurfaceList (Surface [ ] surfaces)

    // modifică buffer-ul din față cu primul la rând buffer din spate
    public Flip ( )

    // avansează buffer-ul din spate la următoarea suprafață
    public Next ( )

    public int FrontBufferIndex { get; set; }

    public int BackBufferIndex { get; set; }

    public Surface GetFrontBuffer ( )
    public Surface GetBackBuffer ( )
    public Surface GetSurface (int surfaceIndex)
}
```

Odată construită, o suprafață şi/sau o listă de suprafețe pot fi ataşate la un obiect al vizualului de suprafață sau la un obiect manager de vizuale.

```
// Creează un vizual de suprafață
public SurfaceDrawingVisual (Surface surface )
public SurfaceDrawingVisual (SurfaceList surfaceList )

// Creează un manager de vizuale împreună cu o țintă de remitere suprafață
public VisualManager (Surface surface )
public VisualManager (SurfaceList surfaceList )
```

În plus, o suprafață poate obține date de la un decodor şi/sau poate trimite datele acesteia la un codor, pentru înscrierea unui format specific de fişier. Suprafețele pot, de asemenea, să recepționeze/trimită date de la /la interfețele de efect. O suprafață poate fi construită pentru orice format de pixeli, din setul complet al tipurilor suportate de formate de

suprafață. Totuşi, pot fi făcute unele reglaje la formatul specificat de pixeli, de exemplu, dacă formatul specificat de pixeli este mai mic de 32 de biți per pixel, atunci formatul va fi avansat la 32 de biți per pixel. Ori de câte ori biții sunt ceruți de la o suprafață din formatul original, suprafața va fi copiată la un buffer al formatului cerut al pixelilor, ce utilizează un filtru de conversie al formatului.

Întorcându-ne la fig. 5, deja un alt vizual este un "HwndVisual" **505**, care poziționează un HWnd copil din Win 32, în graful scenic. Într-un mod şi mai particular, programele lăsate moştenire, încă vor mai lucra prin metoda WM_PAINT (sau asemănătoare), care desenează pe un HWnd copil (sau ceva asemănător), pe baza tehnologiei anterioare de grafică. Pentru a sprijini astfel de programe din noul model de procesare a graficii, "HwndVisual" permite Hwnd-ului să fie conținut într-un graf scenic şi să fie deplasat de îndată ce este repoziționat vizualul părinte, aşa după cum este reprezentat în fig. 10A. Ca un rezultat al limitărilor cu Hwnd-urile existente, totuşi atunci când sunt remise, un Hwnd copil poate să fie numai în partea de sus a altor ferestre şi nu poate fi rotit sau scalat la fel ca alte vizuale descrise mai sus. Este posibilă o decupare, aşa după cum este reprezentată în fig. 10B, unde linia întreruptă indică dreptunghiul afişat al HWnd-ului, ce este decupat pe durata mişcării relative, ținând seama de vizualul părinte al său.

Alte tipuri de vizuale **506** sunt, de asemenea, realizabile, iar modelul obiect prezent este extensibil, pentru a permite altora să fie dezvoltate. De exemplu, aşa după cum este reprezentat în fig. 11, un vizual stratificat **1110** permite unui dezvoltator de aplicații să controleze separat informațiile dintr-un vizual, prin şiruri multiple de date, asigurând o mai fină granularitate a comenzii, privitoare la vizualele ce au un singur şir de date. De remarcat faptul că granularitatea similară a comenzii poate fi realizată prin a avea vizuale tip copil separate (de exemplu, arbore) sub un singur vizual părinte, totuşi acest lucru necesită faptul că, respectiv, codul de program lucrează cu vizuale multiple, care este mai complicat decât a lucra cu un simplu vizual stratificat, ce are indexuri la straturile multiple.

Cu titlul de exemplu, în fig. 11, datele de fundal, datele de conținut, precum şi datele de bordură sunt conținute într-un singur vizual stratificat, însă sunt separate unele de altele, aşa după cum sunt indexate, de către o valoare a stratului, de exemplu, 0, 1, sau, respectiv, 2.Straturile pot fi inserate, inclusiv ataşate la fiecare capăt, şi/sau şterse, împreună cu comanda de stratificare (de exemplu, de la stânga la dreapta, aşa după cum se arată), definind o comandă implicită, Z, pentru display. De notat faptul că, pentru securitate, conținutul copilului precum şi alte date dintr-un vizual stratificat nu pot fi enumerate.

Alte tipuri de vizuale includ vizuale recipient şi vizuale redirecționate HWnd copil, la care conținutul este desenat la un bitmap şi încorporat într-un vizual de suprafață. Vizualele tridimensionale permit o conectare între lumile bidimensionale şi cele tridimensionale, de exemplu, o vizualizare ca la o cameră video este posibilă printr-un vizual bidimensional, ce are o vedere într-o lume tridimensională.

Multe dintre obiectele resurse sunt invariabile odată create, adică odată ce acestea sunt create, acestea nu pot fi modificate din diverse metode, ce includ elemente simplificatoare de pătrundere, împiedicând falsificarea de către alții şi simplificând interacțiunea cu elemente şi cu API-uri. De remarcat faptul că acest lucru simplifică, în general, sistemul. Ar fi de notat, totuşi, faptul că este posibil să ai un sistem unde astfel de obiecte sunt schimbătoare, însă, de exemplu, ar necesita administrarea unui graf de subordonare. De exemplu, în timp ce este posibil de avut un sistem unde astfel de obiecte sunt schimbătoare, dacă codul de program a modificat setarea "clip"-ului la un "Visual", vizualul ar avea nevoie să fie re-remis, în felul acesta, necesitând un mecanism de notificare / înregistrare, de exemplu, dacă un nou "clip" este asignat la un vizual, vizualul se înregistrează el însuşi cu "clip"-ul, pentru notificări (cum ar fi o notificare de modificare clip). Astfel într-o implementare, în scopuri de simplificare, obiectele resursă sunt invariabile.

Aceste obiecte resursă pot fi definite, împreună cu un constructor, care este o cale directă, generică, de a crea un obiect sau de a utiliza un obiect creator însoțitor, aşa cum este descris mai jos. De pildă, pentru a crea un "SolidColorBrush" (pensulă pentru culori uniforme), (obiectele de tip pensulă sunt descrise mai jos), poate fi utilizat un constructor:

Brush MyBrush = new SolidColorBrush (Colors. Red);

Utilizatorul poate, de asemenea, să utilizeze membrii statici de pe clasa de pensule ("Brush"), pentru a obține un set de culori predefinite.

Deoarece obiectele invariabile nu pot fi modificate, pentru a modifica efectiv un obiect, utilizatorul are nevoie să creeze un nou obiect şi să îl înlocuiască pe cel vechi cu acesta. În acest scop, multe dintre obiectele resursă din sistem pot utiliza secvența de creatori, la care obiectele invariabile sunt create împreună cu o clasă de creatori, care este o clasă însoțitoare. care este schimbătoare. Utilizatorul creează un obiect neschimbător, pentru a oglindi setul de parametrii de pe "builder" (creator), creează un nou "builder" pentru acest obiect şi îl inițializează de la obiectul invariabil. Utilizatorul modifică apoi "builder"-ul atât cât este necesar. Odată efectuat acest lucru, utilizatorul poate construi un nou obiect, prin modificarea "builder"-ului şi prin reutilizarea lui pentru a crea un alt obiect invariabil. De observat faptul că este de dorit a avea obiecte invariabile, împreună cu proprietățile de setare, iar aceste obiecte neschimbătoare nu pot fi modificate, ci numai înlocuite prin derularea unui eveniment cu modificare de proprietăți.

În felul acesta, în locul utilizării unui constructor, pentru a crea un "SolidColorBrush", aşa după cum este descris mai sus, poate fi utilizat un "SolidColorBrushBuilder" (creator de pensule de culori uniforme):

```
SolidColorBrushBuilder MyBuilder = new
SolidColorBrushBuilder ();
MyBuilder. Color = Colors. Red;
Brush MyBrush = MyBuilder. ToBrush ();
```

Majoritatea obiectelor, care iau valori statice, pot, de asemenea, să ia obiecte de animație. De pildă, pe "DrawingContext" (contextul de desenare), există o corecție la "DrawCircle" (desenează cerc), care ia un "PointAnimationBase" (indică baza de animație), pentru centrul cercului. În această privință, utilizatorul poate să specifice informația bogată de animație, la nivelul primitivei. Pentru obiectele resursă, acolo există o colecție de animație, în afară de valoarea de bază. Acestea sunt combinate, la care, dacă utilizatorul doreşte să animeze exemplul de mai sus, utilizatorul ar putea să specifice următoarea linie exemplu, înainte ca pensula să fie creată:

```
MyBuilder. ColorAnimations. Add (new ColorAnimation (...));
```

De remarcat că un obiect cu parametrii de animație este încă invariabil, deoarece parametrii lui de animație sunt statici. Totuşi, atunci când graful scenic este procesat (de exemplu, parcurs), semnificația parametrilor de animație se modifică cu timpul, dând impresia de date animate nestatice.

Aşa după cum este descris mai sus, vizualele pot fi determinate prin popularea contextelor lor de desenare cu diverse primitive de desenare, incluzând geometria ("Geometry"), datele de imagine ("ImageData") şi datele video ("VideoData"). În plus, există un set de resurse şi de clase, care sunt distribuite prin acestă întreagă stivă. Acesta include trăgătoare ("Pens"), pensule ("Brushes"), geometrie ("Geometry"), transformate ("Transforms") şi efecte ("Effects"). Contextul "IDrawingContext" etalează un set de operații de desenare, care poate fi utilizat pentru a popula un "DrawingVisual" (vizual de desenare), un "ValidationVisual" (vizual de validare). Contextul "ISurfaceDrawingContext", o interfață de bază pentru desenarea "Idrawing",

a contextului, poate fi utilizat pentru a popula un "SurfaceVisual" (vizual de suprafață). Cu alte cuvinte, contextul de desenare expune un set de operații de desenare; pentru fiecare operație de desenare există două metode, una care ia constantele drept argumente şi una care ia animatorii drept argumente.

Metoda "DrawLine" (trasează linii) trasează o linie cu trăgătorul specific, de la punctul de start la punctul de sfârşit.

```
public  void  DrawLine (Pen  pen,  Point  start,  Point  end);
public  void  DrawLine (
      Pen  pen,
      PointAnimationBase  start,
      PointAnimationBase  end);
```

Metoda "DrawRoundedRectangle" (desenează dreptunghi cu colțuri rotunjite) desenează un dreptunghi cu colțuri rotunjite, cu pensula şi penița specificate; pensula şi penița pot fi zero.

```
      public  void  DrawRoundedRectangle (
            Brush  brush,
            Pen  pen,
            Point  topLeft,
            Size  size,
            float  radius );
      public  void  DrawRoundedRectangle (
            Brush  brush,
            Pen  pen,
            PointAnimationBase  topLeft,
            SizeAnimationBase  size,
            NumberAnimationBase  radius);
      public  void  DrawRoundedRectangle (
            Brush  brush,
            Pen  pen,
            Point  topLeft,
            Point  bottomRight,
            float  rx,
            float  ry);
      public  void  DrawRoundedRectangle (
            Brush  brush,
            Pen  pen,
            PointAnimationBase  topLeft,
            PointAnimationBase  bottomRight,
            NumberAnimationBase  radiusX,
            NumberAnimationBase  radiusY)
```

Metoda "DrawGeometry" (desenează geometrie) desenează o traiectorie cu ajutorul pensulei şi a peniței specificate; pensula şi penița pot fi zero.

```
      public  void  DrawGeometry (
            Brush  brush,
            Pen  pen,
            Geometry  geometry);
```

Metoda "DrawRectangle" (desenează dreptunghi) desenează un dreptunghi cu ajutorul pensulei şi a peniței specificate; pensula şi penița pot fi zero.

```
public  void  DrawRectangle (
    Brush  brush,
    Pen pen,
    Point  topLeft,
    Size  size);
public void  DrawRectangle (
    Brush  brush,
    Pen  pen,
    Point  AnimationBase  topLeft,
    SizeAnimationBase  size);
```

Metoda "DrawSurface" (desenează suprafețe) desenează o suprafață.

```
public  void  DrawSurface (
    Surface  surface,
    Point  topLeft,
    Size  size,
    float  opacity);
public  void  DrawSurface (
    Surface  image,
    PointAnimationBase  topLeft,
    SizeAnimationBase  size,
    NumberAnimationBase  opacity);
```

Geometria este un tip de clasă (fig. 12) care defineşte un schelet de grafică vectorială, fără mişcarea condeiului sau fără umplere. Fiecare obiect de geometrie este de o formă simplă ("LineGeometry"- geometria liniei, "EllipseGeometry"- geometria elipsei, "RectangleGeometry" - geometria dreptunghiului), cu o singură formă complexă ("PathGeometry" - geometria traiectoriei), sau cu o listă "GeometryList" (listă de geometrie) de astfel de forme geometrice, specificate, împreună cu o operație de combinare (de exemplu, reuniune, intersecție, şi aşa mai departe). Aceste obiecte formează o ierarhie de clasă, aşa după cum este reprezentată în fig. 12.

Aşa cum este reprezentată în fig. 13, geometria traiectoriei ("PathGeometry") este o colecție de obiecte "figure" (figură). Fiecare dintre obiectele "figure" sunt compuse, pe rând, dintr-unul sau mai multe obiecte "Segment", care defineşte în realitate forma figurii. O figură ("figure") este o subsecțiune a unei geometrii ("Geometry") care defineşte o colecție de segmente. Această colecție de segmente este o singură serie conectată de obiecte bidimensionale ale lui "Segment". "figure" poate fi fie de o formă închisă cu o arie definită, fie doar o serie conectată de segmente ("Segments") care definesc o curbă, dar nu au o suprafață închisă.

Suprafața umplută din "PathGeometry" (geometria traiectoriei) este definită prin preluarea figurilor ("figures") conținute, care au proprietatea lor de umplere ("Filled") setată pe "true" (adevărat) şi care aplică un "FillMode" (mod de umplere), pentru a determina aria închisă. De remarcat faptul că enumerarea din "FillMode" specifică modul cum suprafețele de intersecție ale obiectelor "figure" (figură) conținute într-o "Geometry" (geometrie) sunt combinate, pentru a forma aria rezultată a lui "Geometry". O regulă de tip "alternează" ("Alternate") determină dacă

un punct este înăuntrul prelatei, prin desenarea, într-un mod conceptual, a unei raze de la acel punct la infinit, în orice direcție, iar apoi, prin examinarea locurilor unde un segment al conformației traversează raza. Prin startarea cu o numărare de la zero şi prin adăugarea de unu la fiecare moment în care un "Segment" traversează raza de la stânga la dreapta şi prin scăderea cu unu, de fiecare dată, când un segment al traiectoriei traversează raza de la dreapta la stânga, după numărarea traversărilor, dacă rezultatul este zero, atunci punctul este în afara traiectoriei. Altfel, acesta este înăuntrul traiectoriei. O regulă "întortocheată" determină dacă un punct de pe acoperământ este înăuntru şi lucrează prin desenarea, în mod conceptual, a unei raze, de la acest punct la infinit, pe orice direcție, şi prin numărarea numărului de segmente ("Segments") ale căii, de la conformația dată pe care o traversează raza. Dacă acest număr este impar, punctul este în interior, dacă este par, punctul este în afară.

Aşa după cum este reprezentat în fig. 14, dacă este desenată geometria (de exemplu, un dreptunghi), poate fi specificată o pensulă sau un trăgător, aşa cum este descris mai jos. În plus, obiectul trăgător are, de asemenea, un obiect pensulă. Un obiect pensulă defineşte felul cum să se umple grafic un plan, şi există o ierarhie de clasă a obiectelor pensule. Acesta este reprezentat în fig. 14, prin dreptunghiul plin, **1402**, care rezultă atunci când este procesat vizualul ce include instrucțiunile şi parametrii pentru dreptunghi şi pentru pensulă.

Aşa după cum este descris mai jos, unele tipuri de pensule ("Brushes", cum ar fi gradienții şi grilele de nouă linii) se dimensionează ele însele. Atunci când se folosesc, dimensiunea pentru aceste pensule este obținută dintr-o căsuță de delimitare, de exemplu, dacă "GradientUnits/DestinationUnits" (unități de gradient/destinație) pentru pensulă ("Brush") este setat la "ObjectBoundingBox" (căsuță de delimitare a obiectului), este folosită căsuța de delimitare a primitivei care este desenată. Dacă aceste proprietăți sunt setate la "UserSpaceOnUse" (spațiu utilizator în uz), atunci este folosit spațiul de coordonate.

Un obiect de tip "Pen" (trăgător) aşteaptă un "Brush" (pensulă), împreună cu proprietățile pentru "Width" (lățime), "LineJoin" (alăturare la linie), "LineCap" (cap de linie), "MiterLimit" (limită de îmbinare în unghi ascuțit), "DashArray" (rețea de haşurare) şi "DashOffset" (deviație de haşurare), aşa cum sunt reprezentate în exemplul de mai jos:

```
public  enum  System. Windows. Media. PenLineCap
{
    Butt,  Round,  Square
}

public  enum  System. Windows. Media. PenLineJoin
{
    Miter,  Round,  Bevel
}

public  class  System. Windows. Media. Pen
{

    / / Constructori
    public  Pen (Color  color,  float  width);
    public  Pen (Brush  brush,  float  width);

    / / Proprietăți
    public  float [ ] DashArray  { get;  }
```

```
        public float DashOffset { get; }
        public FloatAnimationCollection DashOffsetAnimations { get; }

        public PenLineCap LineCap { get; }
        public PenLineJoin LineJoin { get; }

        public float MiterLimit { get; }
        public FloatAnimationCollection MiterLimitAnimations { get; }

        public float Opacity { get; }
        public FloatAnimationCollection OpacityAnimations { get; }

        public Brush Brush { get; }

        public float Width { get; }
        public FloatAnimationCollection WidthAnimations { get; }
    }

    public sealed class System. Windows. Media. PenBuilder : Builder
    {

        / / Câmpuri

        / / Constructori
        public PenBuilder ( );
        public PenBuilder (Color color);
        public PenBuilder (Brush brush);
        public PenBuilder (Pen pen);

        / / Proprietăți
        public float [ ] DashArray { get; set; }

        public float DashOffset { get; set; }
        public FloatAnimationCollectionBuilder DashOffsetAnimations {
          get; }

        public PenLineCap LineCap { get; set; }
        public PenLineJoin LineJoin { get; set; }

        public float MiterLimit { get; set; }
        public FloatAnimationCollectionBuilder MiterLimitAnimations {
          get; }

        public float Opacity { get; set; }
        public FloatAnimationCollectionBuilder OpacityAnimations { get;
          }
```

```
    public Brush Brush { get; set; }

    public float Width { get; set; }
    public FloatAnimationCollectionBuilder WidthAnimations { get; }

    // Metode
    public Pen ToPen ( );
}
```

Aşa cum este menționat mai sus, modelul obiect al graficii din prezenta invenție include un model obiect "Brush" (pensulă), care este, în general, direcționat către conceptul de acoperire a unui plan cu pixeli. Exemple de tipuri de pensule sunt reprezentate în cadrul ierarhiei din fig. 15, iar sub o clasă de bază "Brush" (de pensule), se includ "SolidColorBrush" (pensula cu culoare uniformă), "GradientBrush" (pensula gradient), "ImageBrush" (pensula imagine), "VisualBrush" (pensula vizual) (care poate fi atribuită ca un "Visual" (vizual)), precum şi "NineGridBrush" (pensulă cu grilă de nouă linii). "GradientBrush" include obiectele de "LinearGradient" (gradient liniar) şi de "RadialGradient" (gradient radial). Aşa cum este descris mai sus, obiectele de tip "Brush" sunt invariabile.

```
public abstract class System. Windows. Media. Brush
{
    float Opacity { get; }
    FloatAnimationCollection OpacityAnimations { get; }
}
```

Cele ce urmează, prezintă o clasă exemplu de "BrushBuilder" (creator de pensule):

```
public abstract class System. Windows. Media. BrushBuilder : Builder
{
    public virtual Brush ToBrush ( );
    public override seled object CreateInstance ( );
    {
        return ToBrush ( );
    }
    float Opacity { get; set; }
    FloatAnimationCollectionBuilder OpacityAnimations { get; }
}
```

De notat faptul că obiectele de tip "Brush" (pensulă) pot recunoaşte felul cum acestea se raportează la sistemul de coordonate, atunci când sunt utilizate, şi/sau felul cum acestea se raportează la căsuța de delimitare a formei pe care acestea le utilizează. În general, informații, cum ar fi dimensiunea, pot fi deduse din obiectul pe care îl desenează pensula. Într-un mod mai particular, multe tipuri de pensule utilizează un sistem de coordonate pentru specificarea câtorva dintre parametrii acestora. Acest sistem de coordonate poate fie să fie definit drept

relativ la simpla căsuță de delimitare a formei la care este aplicată pensula, fie poate fi relativ la spațiul de coordonate care este activ la momentul când este utilizată pensula. Acestea sunt cunoscute, drept mod de "ObjectBoundingBox" (căsuță de delimitare obiect) şi, respectiv, de "UserSpaceOnUse" (spațiu utilizator de utilizare).

```
public  enum  System. Windows. Media. BrushMappingMode
{
    ObjectBoundingBox,
    UserSpaceOnUse,
}
```

Un obiect de tip "SolidColorBrush" (pensulă cu culori uniforme) umple, planul identificat, cu o culoare uniformă. Dacă există o componentă "alfa" a culorii, aceasta este combinată într-un mod multiplicativ, împreună cu atributul de opacitate corespunzător, din clasa de bază "Brush". Cele ce urmează, prezintă un obiect exemplu de "SolidColorBrush" (pensulă cu culoare uniformă):

```
public  sealed  class  System. Windows. Media. SolidColorBrush :  Brush
{

    / /  Constructori
    public  SolidColorBrush ( );  / /  inițializează pe negru
    public  SolidColorBrush (Color  color);
    public  SolidColorBrush (System. Windows. Media. Animation. ColorComposer
      colorComposer);

    / /  Proprietăți
    public  Color  Color  { get;  }
    public  IEnumerator  ColorAnimations  { get;  }
}

public  class  System. Windows. Media. SolidColorBrushBuilder: BrushBuilder
{
    / /  Constructori
    public  SolidColorBrushBuilder ( );
    public  SolidColorBrushBuilder (Color  color);
    public  SolidColorBrushBuilder (SolidColorBrush  scp);

    / /  Proprietăți
    public  Color  Color  { get;  set; }
    public  AnimationList  ColorAnimations  { get; }

    / /  Metode
    public  virtual  Brush  ToBrush ( );
}
```

Obiectele "GradientBrush" (pensulă gradient) sau simplii gradienți asigură o umplere în gradient şi sunt desenate prin specificarea unui set de opriri ale gradientului, care specifică culorile de-a lungul câtorva feluri de progresii. Gradientul este desenat prin efectuarea de interpolări liniare între capetele gradientului dintr-un spațiu color RGB cu factorul de contrast de 2,2; interpolarea prin alți factori de contrast sau alte spații color (HSB, CMYK şi aşa mai departe) este, de asemenea, o alternativă realizabilă. Două tipuri de obiecte în gradient includ gradienți liniari şi radiali.

În general, gradienții sunt compuşi dintr-o listă de capete de gradient. Fiecare dintre aceste capete de gradient conține o culoare (împreună cu valoarea alfa inclusă) şi un "offset" (abatere de culoare). Dacă nu este specificat niciun capăt al gradientului, pensula este desenată ca ceva negru, uniform transparent, deoarece nu a fost specificată deloc nicio pensulă. Dacă este specificat numai un capăt al gradientului, pensula este desenată cu o culoare uniformă, având una dintre culorile specificate. Asemănător cu alte clase de resurse, clasa cu capete de gradient (exemplul din tabelul de mai jos) este invariabilă.

```
public class System. Windows. Media. GradientStop
{
    public GradientStop (Color color, float offset);

    public Color Color { get; }
    public AnimationEnumerator ColorAnimations { get; }
    public float Offset { get; }
    public AnimationEnumerator OffsetAnimations { get; }
}

public class System. Windows. Media. GradientStopBuilder : Builder
{
    public GradientStopBuilder ( );
    public GradientStopBuilder (Color color, float offset);

    public Color Color { get; set; }
    public AnimationList ColorAnimations { get; }
    public float Offset { get; set; }
    public AnimationList OffsetAnimations { get; }
    public GradientStop ToGradientStop ( );
}
```

Există, de asemenea, o clasă de colecții, aşa după cum se porneşte din următorul exemplu:

```
public class System. Windows. Media. GradientStopCollection : ICollection
{
    public GradientStopCollection ( ); / / goleşte lista
    public GradientStopCollection (GradientStop [ ] GradientStops);
    public GradientStopCollection (ICollection c);
```

```
        // IEnumerable
        public IEnumerator GetEnumerator ( );

        // ICollection
        public void CopyTo (Array array, int index);
        public bool ICollection. IsSynchronized { get { return false; } }
        public int Count { get; }
        public object ICollection. SyncRoot { get; }

        // Funcții suplimentare
        public GradientStop this [int index] { get; }
        public bool Contains (GradientStop value);
        public int IndexOf (GradientStop value); // returnează pe primul
        public int IndexOf (GradientStop value, int startIndex);
        public int IndexOf (GradientStop value, int startIndex, int count);
        public int LastIndexOf (GradientStop value);
        public int LastIndexOf (GradientStop value, int startIndex);
        public int LastIndexOf (GradientStop value, int startIndex, int count);
        public GradientStopCollection GetRange (int index, int count);
    }

    public class System. Windows. Media. GradientStopCollectionBuilder: Builder,
        IList
    {
        public GradientStopCollectionBuilder ( );
        public GradientStopCollectionBuilder (GradientStop [ ] GradientStops);
        public GradientStopCollectionBuilder (ICollection c);
        public GradientStopCollectionBuilder (GradientStopCollection
            GradientStops);

        // IEnumerable
        public IEnumerator GetEnumerator ( );

        // ICollection
        public void CopyTo (Array array, int index);
        public bool ICollection. IsSynchronized { get { return false; } }
        public int Count { get; }
        public object ICollection. SyncRoot { get; }

        // IList
        public bool IsFixedSize { get { return false; } }
        public bool IsReadOnly { get { return false; } }
        public object IList. this [int index] { get; set; }
        public int IList. Add (object value);
        public void Clear ( );
        public bool IList. Contains (object value);
        public int IList. IndexOf (object value); // returnează pe primul
```

```
    public void Ilist. Insert (int index, object value);
    public void Ilist. Remove (object value); / / transferă-l pe primul
    public void RemoveAt (int index);

    / / Funcții suplimentare
    public GradientStop this [int index] { get; set; }
    public int Add (GradientStop value);
    public bool Contains (GradientStop value);
    public int IndexOf (GradientStop value); / / returnează pe primul
    public int IndexOf (GradientStop value, int startIndex);
    public int IndexOf (GradientStop value, int startIndex, int count);
    public int LastIndexOf (GradientStop value);
    public int LastIndexOf (GradientStop value, int startIndex);
    public int LastIndexOf (GradientStop value, int startIndex, int count);
    public void Insert (int index, GradientStop value);
    public void Remove (GradientStop value); / / transferă-l pe primul
    public void AddRange (ICollection c);
    public void InsertRange (int index, ICollection c);
    public void RemoveRange (int index, int count);
    public void SetRange (int index, ICollection c);
    public GradientStopCollectionBuilder GetRange (int index, int count);

    / / "Capacity" (Capacitatea) este o sugestie. Se va lansa o excepție dacăaceasta
este
       setată la o valoare mai mică decât "Count".
    public int Capacity { get; set; }

    / / Supraîncărcările lui "Builder" (creatorului)
    public override object Build ( );
    public override void ResetBuilder ( );
    public override void SetBuilder (Object example );

    public GradientStopCollection ToGradientStopCollection ( );
}
```

Aşa după cum este reprezentată în tabelul de mai jos, metoda "GradientSpreadMethod" (metodă de distribuire a gradientului) specifică felul cum gradientul trebuie să fie desenat în afara vectorului sau a spațiului specificat. Există trei valori, inclusiv "pad"-ul (umplutura), la care culorile din margine (prima şi ultima) sunt folosite pentru a umple spațiul rămas, pentru a oglindi, în care dintre capete sunt, în mod repetat, reproduse în ordine inversă, pentru a umple spațiul şi pentru a repeta, în care dintre capete, sunt repetate în ordine până ce spațiul este umplut:

```
public enum System. Windows. Media. GradientSpreadMethod
{
    Pad,
    Reflect,
    Repeat
}
```

Fig. 16 arată exemple ale metodei "GradientSpreadMethod". Fiecare conformație are un gradient liniar, mergând de la alb la gri. Linia uniformă reprezintă vectorul gradient.

"LinearGradient" (gradientul liniar) specifică o pensulă de gradient liniar de-a lungul unui vector. Capetele individuale specifică capete de culori de-a lungul acestui vector. Un exemplu este prezentat în tabelul de mai jos:

```
public class System. Windows. Media. LinearGradient : GradientBrush
{
    / / Pune un gradient cu două culori precum şi un vector gradient
    / / specificați pentru a umple obiectul la care este folosit gradientul.
    / / Acest lucru înseamnă "ObjectBoundingBox" (căsuță de delimitare a
    / / obiectului) pentru proprietatea de "GradientUnits" (unități de gradient)
    public LinearGradient (Color color1, Color color2, float angle);

    public BrushMappingMode GradientUnits { get; }
    public Transform GradientTransform { get; }
    public GradientSpreadMethod SpreadMethod { get; }

    / / Vector de gradient
    public PointVectorStart { get; }
    public PointAnimationCollection VectorStartAnimations { get; }
    public Point VectorEnd { get; }
    public PointAnimationCollection VectorEndAnimations { get; }

    / / Capetele gradientului
    public GradientStopCollection GradientStops { get; }
}


public class System. Windows. Media. LinearGradientBuilder:
      GradientBrushBuilder
{
    public LinearGradientBuilder ( );
    public LinearGradientBuilder (Color color1, Color color2, float angle);
    public LinearGradientBuilder (LinearGradient lg);

    / / "GradientUnits"(unități de gradient): Referința este "ObjectBoundingBox"
    public BrushMappingMode GradientUnits { get; set; }
    / / "GadientTransform"(transformata gradientului); Referința este identiatea
    public Transform GradientTransform { get; set; }
    / / "SpreadMethod" (metoda de distribuire): Referința este "Pad"
    public GradientSpreadMethod SpreadMethod { get; set; }

    / / Vector de gradient
    / / Vector de referință este (0,0) - (1,0)
    public Point VectorStart { get; set; }
    public PointAnimationCollectionBuilder VectorStartAnimations { get;
      set; }
```

```
    public PointVectorEnd { get; set; }
    public PointAnimationCollectionBuilder VectorEndAnimations { get; set;
      }

    // Capetele gradientului
    public void AddStop (Color color, float offset);
    public GradientStopCollectionBuilder GradientStops { get; set; }
}
```

"RadialGradient"-ul (gradientul radial) este similar la modelul de programare referitor la gradientul liniar. Totuşi, având în vedere că gradientul liniar are un punct de început şi unul de sfârşit, pentru a defini vectorul gradient, gradientul radial are un cerc, împreună cu un punct focal, pentru a defini comportarea de gradient. Cercul defineşte punctul de sfârşit al gradientului, adică un capăt al gradientului la 1,0 defineşte culoarea la cerc. Punctul focal defineşte centrul gradientului. Un capăt al gradientului la 0,0 defineşte culoarea la punctul focal.

Fig. 17 prezintă un gradient radial, care este de la alb la gri. Cercul din exterior reprezintă cercul gradient, în timp ce punctul desemnează punctul focal. Acest gradient exemplu are "SpreadMethod"-ul (metoda de distribuire) setat la "Pad":

```
public class System. Windows. Media. RadialGradient : GradientBrush
{
    // Pune un gradient cu două culori.
    // Acest lucru înseamnă "ObjectBoundingBox" pentru proprietatea de
    // "GradientUnits" împreună cu un centru la (0.5, 0.5)
    // o rază a cercului de 0.5 şi un punct focal la (0.5, 0.5)
    public RadialGradient (Color color1, Color color2);

    public BrushMappingMode GradientUnits { get; }
    public Transform GradientTransform { get; }
    public GradientSpreadMethod SpreadMethod { get; }

    // Definire gradient
    public Point CircleCenter { get; }
    public PointAnimationCollection CircleCenterAnimations { get; }
    public float CircleRadius { get; }
    public FloatAnimationCollection CircleRadiusAnimations { get; }
    public Point Focus { get; }
    public PointAnimationCollection FocusAnimations { get; }

    // Capetele gradientului
    public GradientStopCollection GradientStops { get; }
}

public class System. Windows. Media. RadialGradientBuilder:
      GradientBrushBuilder
{
    public RadialGradientBuilder ( );
```

```
    public RadialGradient (Color color1, Color color2);
    public RadialGradientBuilder (RadialGradient rg);

    // "GradientUnits": Referința este "ObjectBoundingBox"
    public BrushMappingMode GradientUnits { get; set; }
    // "GradientTransform": Referința este identitatea
    public Transform GradientTransform { get; set; }
    // "SpreadMethod": Referința este "Pad"
    public GradientSpreadMethod SpreadMethod { get; set; }

    // Definire gradient
    public Point CircleCenter { get; set; } // Referința: (0.5, 0.5)
    public PointAnimationCollectionBuilder CircleCenterAnimations { get;
      set; }
    public float CircleRadius { get; set; } // Referința: 0.5
    public FloatAnimationCollectionBuilder CircleRadiusAnimations {
      get; set; }
    public Point Focus { get; set; } // Referința: (0.5, 0.5)
    public PointAnimationCollectionBuilder FocusAnimations { get; set; }

    // Capetele gradientului
    public void AddStop (Color color, float offset);
    public GradientStopCollectionBuilder GradientStops { get; set; }
}
```

Un alt obiect pensulă, reprezentat în fig. 15, este un obiect "VisualBrush" (pensulă de vizual). În mod conceptual, "VisualBrush" asigură o cale de a avea un vizual desenat într-o manieră repetată, învelit ca țiglele, drept umplutură. Obiectele de pictare a vizualelor prevăd, de asemenea, un mecanism pentru limbajul de marcare, pentru a lucra direct, cu stratul API, la un nivel de resurse, aşa cum este descris mai jos. Un exemplu de astfel de umplere este reprezentat, în fig. 14, de către pensula de vizuale, ce completează un vizual (precum şi orice vizual copil), care specifică o singură conformație circulară **1420**, cu această conformație circulară umplând un dreptunghi **1422**. Astfel, obiectul "VisualBrush" poate completa un vizual, pentru a defini felul cum va fi desenată această pensulă, care introduce un tip de multiple utilizări, pentru vizuale. În acest mod, un program poate utiliza o grafică "metafile" arbitrară, pentru a umple o suprafață, cu o pensulă sau cu un trăgător. Deoarece aceasta este o formă comprimată, pentru stocarea şi utilizarea graficii arbitrare, aceasta deserveşte o resursă a graficii. Cele ce urmează, prezintă un obiect exemplu de "VisualBrush" (pensulă pentru vizuale):

```
public class System. Windows. Media. VisualBrush: Brush
{
    public VisualBrush (Visual v);

    public BrushMappingMode DestinationUnits { get; }
    public BrushMappingMode ContentUnits { get; }
    public Transform Transform { get; }
```

```
    public Rect ViewBox { get; }
    public Stretch Stretch { get; }
    public HorizontalAlign HorizontalAlign { get; }
    public VerticalAlign VerticalAlign { get; }

    public Point Origin { get; }
    public PointAnimationCollection OriginAnimations { get; }
    public Size Size { get; }
    public SizeAnimationCollection SizeAnimations { get; }

    // "Visual" (vizualul)
    public Visual Visual { get; }
}

public class System.Windows.Media.VisualBrushBuilder: BrushBuilder
{
    public VisualBrushBuilder ( );
    public VisualBrushBuilder (Visual v);
    public VisualBrushBuilder (VisualBrush vb);

    // "DestinationUnits"(unităţi destinaţie):Referinţa este "ObjectBoundingBox"
    public BrushMappingMode DestinationUnits { get; set; }
    // "ContentUnits" (unităţi conţinut): Referinţa este "ObjectBoundingBox"
    public BrushMappingMode ContentUnits { get; set; }
    // Transformata: Referinţa este "Identity" (identitatea)
    public Transform Transform { get; set; }
    // "ViewBox": Referinţa este (0, 0, 0, 0) – nesetat şi ignorat
    public Rect ViewBox { get; set; }
    // "Stretch" (extindere): Referinţa este "None" (nici una) -- şi este ignorată
    //              deoarece "ViewBox"-ul nu este setat
    public Stretch Stretch { get; set; }
    // "HorizontalAlign" (aliniere orizontală): Referinţa este "Center" (centru) şi
    //              este ignorată
    public HorizontalAlign HorizontalAlign { get; set; }
    // "VerticalAlign"(aliniere verticală):Referinţa este "Center" şi este ignorată
    public VerticalAlign VerticalAlign { get; set; }

    // "Origin" (originea): Referinţa este (0, 0)
    public Point Origin { get; set; }
    public PointAnimationCollectionBuilder OriginAnimations { get; set; }
    // "Size" (dimensiunea): Referinţa este (1, 1)
    public Size Size { get; set; }
    public SizeAnimationCollectionBuilder SizeAnimations { get; set; }

    // "Visual" (vizualul): Referinţa este zero -- nu este desenat nimic
    public Visual Visual { get; set; }
}
```

Conținutul lui "VisualBrush" nu are nicio suprafață de separație intrinsecă şi, efectiv, descrie un plan infinit. Acest conținut există în spațiul lui propriu de coordonate, iar spațiul, care este umplut de către "VisualBrush", este spațiul local de coordonate la momentul aplicației. Spațiul de conținut este cartografiat în spațiul local, pe baza proprietăților de "ViewBox" (căsuța de vizualizare), "ViewPort" (port de vizualizare), "Alignments" (alinieri) şi de "Stretch" (extindere). "ViewBox"-ul este specificat în spațiul de conținut, iar acest dreptunghi este cartografiat în dreptunghiul "ViewPort" (aşa cum este specificat prin proprietățile de "Origin" (origine) şi de "Size" (dimensiune)).

"ViewPort"-ul defineşte locația unde conținutul va fi eventual desenat, creând învelişul de țigle de bază, pentru acest "Brush" (pensulă). Dacă valoarea lui "DestinationUnits" (unități destinație) este "UserSpaceOnUse" (spațiu utilizator în uz), proprietățile de "Origin" (origine) şi de "Size" (dimensiune) sunt considerate a fi în spațiul local la momentul aplicației. Dacă în locul valorii lui "DestinationUnits", este "ObjectBoundingBox" (căsuța de delimitare obiect), atunci un "Origin" şi un "Size" sunt considerate a fi în spațiul de coordonate, unde 0,0 este colțul de sus/stânga al căsuței de delimitare a obiectului ce este trasat cu pensula, iar 1,1 este colțul de jos/dreapta al aceleiaşi căsuțe. De exemplu, să considerăm un "RectangleGeometry" (geometrie a dreptunghiului) ce este umplut, care este desenat de la 100, 100 la 200, 200. Într-un astfel de exemplu, dacă "DestinationUnits" este "UserSpaceOnUse", un "Origin" de 100, 100 şi un "Size" de 100, 100 ar descrie întreaga suprafață a conținutului. Dacă "DestinationUnits" este "ObjectBoundingBox", un "Origin" de 0, 0 şi un "Size" de 1, 1 ar descrie întreaga suprafață a conținutului. Dacă "Size" este gol, acest "Brush" nu remite nimic.

"ViewBox"-ul este specificat în spațiul conținutului. Acest dreptunghi este transformat, pentru a se potrivi în interiorul "ViewPort"-ului, aşa cum este determinat de către proprietățile de "Alignment" şi de proprietatea de "Stretch". Dacă în "Stretch" nu este nimic, atunci nu este aplicată nicio scalare la conținut. Dacă "Stretch" este "Fill" (umplut), atunci "ViewBox"-ul este scalat în mod independent atât pe X, cât şi pe Y, pentru a fi de aceeaşi dimensiune cu "ViewPort"-ul. Dacă "Stretch" este "Uniform" sau "UniformToFill" (uniform de umplut), logica este asemănătoare, însă dimensiunile X şi Y sunt scalate uniform, păstrând raportul aspectelor conținutului. Dacă "Stretch" este "Uniform", "ViewBox"-ul este scalat, pentru a avea dimensiunea mai comprimată, egală cu mărimea "ViewPort"-ului. Dacă "Stretch" este "UniformToFill", "ViewBox"-ul este scalat, pentru a avea dimensiunea mai puțin comprimată, egală cu mărimea "ViewPort"-ului. Cu alte cuvinte, atât "Uniform"-ul, cât şi "UniformToFill", păstrează raportul aspectelor, însă "Uniform"-ul garantează că întregul "ViewBox" este în interiorul "ViewPort"-ului (porțiuni virtual de părăsire a "ViewPort"-ului neacoperit de "ViewBox"), iar "UniformToFill" garantează că întregul "ViewPort" este umplut de către "ViewBox" (porțiuni virtual cauzatoare ale "ViewBox"-ului de a fi în afara "ViewPort-ului). Dacă "ViewBox"-ul este gol, atunci nu se va folosi "Stretch"-ul. De remarcat că aliniamentul încă va apărea, iar acesta va poziționa "ViewBox"-ul "point" (punct).

Fig. 18 asigură reprezentări ale unei singure plăci **1800**, de grafică, remisă, împreună cu diverse setări de extindere, incluzând o placă **800**, atunci când extinderea este setată la "none" (nimic). Placa **1802** este o reprezentare atunci când extinderea este setată la "Uniform", placa **1804**, atunci când extinderea este setată la "UniformToFill", iar placa **1806** atunci când extinderea este setată la "Fill".

Odată ce "ViewPort"-ul este determinat (pe baza lui "DestinationUnits"), iar dimensiunea lui "ViewBox" este determinată (pe baza lui "Stretch"), "ViewBox" -ul are nevoie să fie poziționat în interiorul lui "ViewPort". Dacă "ViewBox"-ul este de aceeaşi mărime ca "ViewPort"-ul (dacă "Stretch" este "Fill" sau dacă se întâmplă doar să apară împreună cu una dintre celelalte trei valori ale lui "Stretch"), atunci "ViewBox"-ul este poziționat la "Origin", astfel ca să fie identic cu

"ViewPort"-ul. Altfel, se iau în considerare "HorizontalAlignment" (alinierea orizontală) şi "VerticalAlignment" (alinierea verticală). Pe baza acestor proprietăţi, "ViewBox"-ul este aliniat la ambele dimensiuni X şi Y. Dacă "HorizontalAlignment" este "Left" (la stânga), atunci marginea stângă a lui "ViewBox" va fi poziţionată la marginea "Left" (stângă) a lui "ViewPort". Dacă acesta este "Center" (centru), atunci centrul lui "ViewBox" va fi poziţionat la centrul lui "ViewPort", iar dacă este "Right" (dreapta), atunci se vor întâlni marginile din dreapta. Procesul este repetat pentru dimensiunea Y.

Dacă "ViewBox"-ul este (0, 0, 0, 0), acesta este considerat nesetat, drept care "ContentUnits" (unităţile conţinutului) sunt luate în considerare. Dacă "ContentUnits" sunt "UserSpaceOnUse", nu apare nicio scalare sau niciun decalaj, iar conţinutul este desenat în "ViewPort", fără nicio transformată. Dacă "ContentUnits" sunt "ObjectBoundingBox", atunci originea conţinutului este aliniată, împreună cu "ViewPort Origin" (originea portului de vizualizare), iar conţinutul este scalat de către lăţimea si înălţimea căsuţei de delimitare a obiectului.

Atunci când se umple un spaţiu cu un "VisualBrush", conţinutul este cartografiat în "ViewPort", ca mai sus, şi este decupat la "ViewPort". Acesta formează placa de bază pentru umplere, iar ceea ce rămâne din spaţiu este umplut pe baza lui "TileMode" (modul de acoperire cu ţiglă) al lui "Brush". În concluzie, dacă este setat, este aplicată transformata lui "Brush"- aceasta apare după toate celelalte cartografieri, scalari, decalări etc.

Enumerarea din "TileMode" (modul tip placă ţiglă) este utilizată pentru a descrie dacă şi cum este umplut un spaţiu de către "Brush"-ul lui. Un "Brush", care poate fi acoperit cu plăci, are definit un dreptunghi tip placă, iar această placă are o locaţie de bază în interiorul spaţiului ce este umplut. Restul spaţiului este umplut pe baza valorii lui "TileMode". Fig. 19 asigură o reprezentare a graficii exemplu, împreună cu diverse setări ale lui "TileMode", ce includ "None" **1900**, "Tile" **1092**, "FlipX" **1904**, "FlipY" **1906** şi "FlipXY" **1908**. Placa cea mai de sus-stânga din diversele grafici exemplu conţine placa de bază.

Fig. 20 reprezintă un proces pentru generarea pixelilor pentru această pensulă. De notat faptul că logica descrisă în fig. 20 este numai o singură cale posibilă pentru a implementa logica, iar aceasta ar trebui înţeleasă prin faptul că şi celelalte căi, incluzând căi mai eficiente, sunt realizabile. De exemplu, există probabil căi mai eficiente de procesare a datelor, de pildă, astfel încât conţinutul să nu fie desenat pentru fiecare repetiţie, cu ţigla (placa) desenată şi memorată ascuns. Totuşi, fig. 20 asigură o prezentare directă.

În general, de fiecare dată când este desenat conţinutul secvenţei, un nou sistem de coordonate este creat. Originea şi decalajul fiecărei repetiţii este specificat de către proprietăţile "Origin" şi "Size", aşa cum sunt filtrate prin proprietăţile lui "DestinationUnits" si ale lui "Transform".

Un reper de coordonate este fixat pe baza proprietăţii lui "DestinationUnits". În acest scop, dacă la pasul **2000**, proprietatea lui "DestinationUnits" este "UserSpaceOnUse", reperul curent de coordonate, din momentul când a fost folosită pensula, este reperul de startare de coordonate, prin pasul **2002**. Dacă, în schimb, la pasul **2004**, proprietatea este "ObjectBoundingBox", se foloseşte căsuţa de delimitare a geometriei la care această pensulă este aplicată, aşa cum este reprezentată de pasul **2004**, pentru a seta un nou reper de coordonate, astfel încât colţul din stânga sus al căsuţei de delimitare se poziţionează pe (0, 0), iar colţul de jos stânga al căsuţei de delimitare se poziţionează pe (1, 1). În fiecare dintre cazuri, la pasul **2006**, proprietatea "Transform" este aplicată la acest reper de coordonate, care în fond defineşte un grilaj.

Fig. 21 reprezintă un "Grid " (grilaj) al lui "VisualBrush", care este definit pentru ţiglele dintr-un "VisualBrush". Primul cerc este un simplu grilaj, iar al doilea are un "Transform" (transformată) cu un "Skew" (oblicitate) pe direcţia x, de 47.

La pasul **2008**, vizualul este desenat în fiecare celulă a grilajului, aşa cum este reprezentat în fig. 22, unde vizualul desenează datele specifice. Dacă la pasul **2010**, este specificat un "ViewBox", "Visual"-ul este introdus în celula grilei, aşa cum este specificat de către atributele de "ViewBox", "Stretch", "HorizontalAlign" şi de "VerticalAlign", prin pasul **2012**. Proprietăţile de "DestinationUnits" şi de "Transform" sunt utilizate pentru a aplica transformata corectă, astfel ca vizualul să se alipească în căsuţa grilei.

Dacă nu este specificat niciun "ViewBox", atunci este stabilit un nou sistem de coordonate, pentru desenarea conţinutului de la pasul **2014**.

Reperul de coordonate este setat astfel că originea lui este la punctul "Origin", pentru această celulă particulară a grilei ce este desenată.

La pasul **2018**, este aplicat un "clip" (decupare), pe baza proprietăţii "Size", astfel că această ţiglă nu se va desena în afara delimitărilor celulei. "Origin" şi "Size" sunt modificate în mod adecvat, pe baza proprietăţii lui "DestinationUnits".

Sistemul de coordonate este apoi modificat, pe baza proprietăţii lui "SourceUnits" (unităţi sursă). În acest scop, dacă la pasul **2020**, proprietatea lui "SourceUnits" este "ObjectBoundingBox", transformata specifică de scalare este aplicată la pasul **2026**, altfel aceasta este "UserSpaceOnUse" şi nu este aplicată nicio nouă transformată. Proprietatea lui "Transform" este aplicată la pasul **2024**, iar conţinutul este desenat la pasul **2026**.

De notat faptul că, dacă orice parte a mărimii este zero, nu este desenat nimic, iar dacă "Stretch" este "None", transformata pentru "ViewBox" este fixată, astfel că o unitate din noul reper de coordonate este egală cu o unitate din vechiul reper de coordonate. Transformata devine în mod esenţial un "offset" (decalaj), bazat pe atributele de aliniere şi pe mărimea lui "ViewBox". Aşa după cum este descris mai sus, la paşii **2010** şi **2012**, "Stretch"-ul şi proprietăţile de aliniere se aplică numai dacă este specificat un "ViewBox". "ViewBox"-ul specifică un nou sistem de coordonate pentru conţinut, iar "Stretch"-ul ajută pentru a specifica felul cum acest conţinut este indicat în "ViewBox". Opţiunile de aliniere aliniază "ViewBox"-ul, nu conţinutul. Astfel, de exemplu, dacă "viewbox"-ul (căsuţa de vizualizare) este setat la "0 0 10 10" şi este desenat ceva la -10, -10 şi aliniat la colţul din stânga sus, acest lucru va fi desprins în afară.

Întorcându-ne la fig. 15, pensula imagine poate fi gândită ca un caz special de "VisualBrush". Chiar dacă un program poate crea un vizual, poate pune o imagine în acesta şi poate să-l ataşeze la "VisualBrush", API-ul care să facă acest lucru ar fi incomod. Deoarece nu există niciun reper indispensabil de coordonate ale conţinutului, nu se mai aplică deloc elementele proprietăţilor "ViewBox"-ul şi ale lui "ContentUnits".

```
public class System. Windows. Media. ImageBrush: Brush
{
    public ImageBrush (ImageData image);

    public BrushMappingMode DestinationUnits { get; }
    public Transform Transform { get; }

    public Stretch Stretch { get; }
    public HorizontalAlign HorizontalAlign { get; }
    public VerticalAlign VerticalAlign { get; }

    public Point Origin { get; }
```

```
    public PointAnimationCollection OriginAnimations { get; }
    public Size Size { get; }
    public SizeAnimationCollection SizeAnimations { get; }

    public ImageData ImageData { get; }
}

public class System. Windows. Media. ImageBrushBuilder: BrushBuilder
{
    public ImageBrushBuilder ( );
    public ImageBrushBuilder (ImageData image);
    public ImageBrushBuilder (ImageBrush ib);

    / / "DestinationUnits": Referința este "ObjectBoundingBox"
    public BrushMappingMode DestinationUnits { get; set; }
    / / "Transform" (transformata): Referința este identitatea
    public Transform Transform { get; set; }

    / / "Stretch" (extindere): Referința este "None"
    public Stretch Stretch { get; set; }
    / / "HorizontalAlign": Referința este "Center"
    public HorizontalAlign HorizontalAlign { get; set; }
    / / "VerticalAlign": Referința este "Center"
    public VerticalAlign VerticalAlign { get; set; }

    / / "Origin": Referința este (0, 0)

    public Point Origin { get; set; }
    public PointAnimationCollectionBuilder OriginAnimations { get; set; }

    / / "Size": Referința este (1, 1)
    public Size Size { get; set; }
    public SizeAnimationCollectionBuilder SizeAnimations { get; set; }

    / / "ImageData" (date imagine): Referința este zero -- nu este desenat nimic
    public ImageData ImageData { get; set; }
}
```

"NineGridBrush" (pensula cu nouă grile) este foarte asemănătoare cu "ImageBrush"-ul (pensula pentru imagini), exceptând faptul că imaginea este deformată ca mărime. În esență, "NineGridBrush" poate fi gândit ca un tip clientelă al lui "Stretch", la care anumite părți din imagini se extind, în timp ce altele (de exemplu, bordurile) nu se extind. În felul acesta, în tirnp ce "Size"-ul (mărimea) imaginii din "ImageBrush" va determina o scală simplă, "NineGridBrush"-ul va produce o scală neuniformă, până la mărimea dorită. Unitățile pentru zonele nescalate sunt unități ale utilizatorului, atunci când este aplicată pensula, care înseamnă că "ContentsUnits" (unități ale conținutului) (dacă acesta există pentru "NineGridBrush") ar fi setat la "UserUnitsOnUse". Proprietatea "Transform" a lui "Brush" poate fi utilizată în mod efectiv. De remarcat că elementele bordurii se numără de la marginea imaginii.

În chip de exemplu, fig. 23 reprezintă o imagine cu nouă grile, ce este lărgită de la primul moment, **2302**, la al doilea moment, **2304**, cu patru tipuri de zone. Aşa cum este reprezentat în fig. 23, pentru a menține aceeaşi bordură, zonele marcate cu "a" se extind orizontal, zonele marcate cu "b" se extind vertical, zonele marcate cu "c" se extind orizontal şi vertical, iar zonele marcate cu "d" nu se modifică în mărime.

```
public class System. Windows. Media. NineGridBrush: Brush
{
    public NineGridBrush (ImageData image,
                int LeftBorder, int RightBorder,
                int TopBorder, int BottomBorder);

    public BrushMappingMode DestinationUnits { get; }
    public Transform Transform { get; }

    public Point Origin { get; }
    public PointAnimationCollection OriginAnimations { get; }
    public Size Size { get; }
    public SizeAnimationCollection SizeAnimations { get; }


    public int LeftBorder { get; }
    public int RightBorder { get; }
    public int TopBorder { get; }
    public int BottomBorder { get; }

    public ImageData ImageData { get; }
}

public class System. Windows. Media. NineGridBrushBuilder: BrushBuilder
{
    public NineGridBrushBuilder ( );
    public NineGridBrushBuilder (ImageData image,
                int LeftBorder, int RightBorder,
                int TopBorder, int BottomBorder);
    public NineGridBrushBuilder (NineGridBrush ngb);

    / / "DestinationUnits": Referința este "ObjectBoundingBox"
    public BrushMappingMode DestinationUnits { get; set; }
    / / "Transform" : Referința este identitatea
    public Transform Transform { get; set; }

    / / "Origin": Referința este (0, 0)

    public Point Origin { get; set; }
    public PointAnimationCollectionBuilder OriginAnimations { get;
      set; }
```

```
    //  "Size": Referința este (1, 1)
    public Size Size { get; set; }
    public SizeAnimationCollectionBuilder SizeAnimations { get; set; }

    //  *"Border" (bordură): referința la 0
    public int LeftBorder { get; set; }
    public int RightBorder { get; set; }
    public int TopBorder { get; set; }
    public int BottomBorder { get; set; }

    //  "ImageData": Referința este zero -- nu este desenat nimic
    public ImageData ImageData { get; set; }
}
```

Aşa cum este descris la modul general, modelul obiect de grafică al prezentei invenţii include un model obiect "Transform" (transformată), care include tipurile de transformate reprezentate în ierarhia din fig. 24, sub o clasă de bază a "Transform"-ului. Aceste tipuri diferite de componente, care întregesc o transformată, pot include "TransformList" (lista de transformate), "TranslateTransform" (transformata de translaţie), "RotateTransform" (transformata de rotaţie), "ScaleTransform" (transformata de scală), "SkewTransform" (transformata de oblicitate) şi "MatrixTransform" (transformata matricială). Proprietăţile individuale pot fi animate, de exemplu, un dezvoltator de programe poate să anime proprietatea de "Angle" (unghi) a unui "RotateTransform".

Matricele pentru calculele 2D sunt reprezentate ca o matrice 3x3. Pentru transformatele necesare, numai şase valori sunt necesare în locul unei matrice complete de 3x3. Acestea sunt denumite şi definite după cum urmează.

```
m00  m01  0
m10  m11  0
m20  m21  1
```

Atunci când o matrice este multiplicată cu un punct, aceasta transformă acel punct din noul sistem de coordonate în precedentul sistem de coordonate:

$$[X_{newCoordSys}\ y_{new\ CoordSys}\ 1] \cdot \begin{matrix} m00 & m01 & 0 \\ m10 & m11 & 0 \\ m20 & m21 & 1 \end{matrix} = [X_{oldCoordSys}\ y_{old\ CoordSys}\ 1]$$

Transformatele pot fi incluse unele în altele, la orice nivel. Ori de câte ori o nouă transformată este aplicată, este acelaşi lucru cu post-multiplicarea acesteia la matricea curentă a transformatei:

$$[X_{newCoordSys}\ y_{new\ CoordSys1}] \cdot \begin{matrix} m002 & m012 & 0 \\ m102 & m112 & 0 \\ m202 & m212 & 1 \end{matrix} \cdot \begin{matrix} m001 & m011 & 0 \\ m101 & m111 & 0 \\ m201 & m211 & 1 \end{matrix} = [X_{oldCoordSys}\ y_{old\ CoordSys\ 1}]$$

Majoritatea locurilor din API nu preiau în mod direct un "Matrix" (matrice), dar în schimb folosesc clasa lui "Transform", care suportă animația.

```
public struct System. Windows. Media. Matrix
{
    // Construire şi setare
    public Matrix ( ); // referințe la identitate

    public Matrix (
                double m00, double m01,
                double m10, double m11,
                double m20, double m21;

    // "Identity" (identitate)
    public static readonly Matrix Identity;
    public void SetIdentity ( );
    public bool IsIdentity { get; }

    public static Matrix operator * (Matrix matrix1, Matrix matrix2);
    public static Point operator * (Matrix matrix, Point point);

    // Aceste funcții reinițializează matricea curentă cu
    // matricea specificată a transformatei.
    public void SetTranslation (double dx, double dy);
    public void SetTranslation (Size offset);
    public void SetRotation (double angle); // grade
    public void SetRotation (double angle, Point center); // grade
    public void SetRotationRadians (double angle);
    public void SetRotationRadians (double angle, Point center);
    public void SetScaling (double sx, double sy);
    public void SetScaling (double sx, double sy, Point center);
    public void SetSkewX (double angle ); // grade
    public void SetSkewY (double angle ); // grade
    public void SetSkewXRadians (double angle );
    public void SetSkewYRadians (double angle );

    // Aceste funcții post-multiplică matricea curentă
    // cu transformata specificată
    public void ApplyTranslation (double dx, double dy);
    public void ApplyTranslation (Size offAply);
    public void ApplyRotation (double angle); // grade
    public void ApplyRotation (double angle, Point center); //
      grade
    public void ApplyRotationRadian (double angle);
    public void ApplyRotationRadian (double angle, Point center);
    public void ApplyScaling (double sx, double sy);
```

```
        public void ApplyScaling (double sx, double sy, Point center);
        public void ApplySkewX (double angle ); / / grade
        public void ApplySkewY (double angle ); / / grade
        public void ApplySkewXRadians (double angle );
        public void ApplySkewYRadians (double angle );
        public void ApplyMatrix (Matrix matrix);

        / / Material de inversiune
        public double Determinant { get; }
        public bool IsInvertible { get; }
        public void Invert ( ); / / Lansează "ArgumentException" dacă
            este inversabilă ( "! IsInvertable")
        public static Matrix Invert (Matrix matrix);

        / / Elemente individuale
        public double M00 { get; set; }
        public double M01 { get; set; }
        public double M10 { get; set; }
        public double M11 { get; set; }
        public double M20 { get; set; }
        public double M21 { get; set; }
};
```

În conformitate cu un aspect al prezentei invenții, sunt prevăzute un limbaj de marcare şi un model obiect al elementelor, pentru a permite programelor de utilizator, precum şi uneltelor, să interacționeze cu structura de date **216**, a grafului scenic, fără a necesita cunoştințe specifice asupra detaliilor stratului API **212** (fig. 2). În general, este prevăzut un limbaj de marcare pentru grafica vectorială, care cuprinde un format de interschimbare, împreună cu un format simplu de autorizare pe baza marcatorilor, pentru exprimarea graficii vectoriale printr-un model obiect al elementelor. Prin acest limbaj, poate fi programată marcarea (de exemplu, HTML sau conținutul de tip XML). După aceea, pentru a construi un graf scenic, marcarea este analizată gramatical şi este tradusă în obiecte specifice ale stratului API vizual, care au fost descrise mai sus. La acest nivel mai ridicat de operare, sunt prevăzuți un arbore de elemente, un sistem de proprietăți şi un sistem de prezentatori, pentru a manevra mare parte din complexitate, făcând-o accesibilă pentru designerii de scenă, pentru a proiecta posibile scene complexe.

În general, sistemul de grafică vectorială prevede, de regulă, un set de elemente de conformație, precum şi alte elemente, prevede o integrare cu un sistem general de proprietăți, prevede un sistem de grupare şi de compoziție, precum şi o aproximație bistratificată (nivel element şi nivel resursă), astfel că utilizatorul poate programa într-un mod care corespunde cu nevoile de flexibilitate şi de performanță. În acord cu un aspect al prezentei invenții, modelul obiect al elementelor care are de-a face cu grafica vectorială, se găseşte în corelație cu modelul obiect al grafului scenic. Cu alte cuvinte, sistemul de grafică vectorială şi stratul API de "Visual" împart un set de resurse la nivelul modelului obiect al elementelor, de exemplu, obiectul "Brush" (pensulă) este folosit când se desenează la API-ul de "Visual", iar acesta este, de asemenea, tipul proprietății de completare pe "Shape" (conformație). Astfel, în afară de a avea elemente care se găsesc în corelație cu obiectele grafului scenic, limbajul de marcare împarte un număr de resurse ale primitivelor (de exemplu, pensule, transformate, şi aşa mai departe), cu stratul "Visual API". Sistemul de grafică vectorială expune şi extinde de asemenea capacitățile de animație ale stratului "Visual API", care este împărțit în mare măsură între niveluri.

În plus, aşa după cum este descris mai jos, sistemul de grafică vectorială se poate programa la diferite profiluri sau niveluri, ce includ un nivel de elemente şi un nivel de resurse. La nivelul de elemente, fiecare dintre conformaţiile de desenare este reprezentată ca un element, la acelaşi nivel ca şi restul din elementele programabile dintr-o pagină/ecran. Aceasta înseamnă că respectivele conformaţii interacţionează în întregime cu sistemul prezentator, cu evenimentele şi cu proprietăţile. La nivelul de resurse, sistemul de grafică vectorială funcţionează într-un format pur de resurse, similar cu un "metafile" (meta-fişier) al graficii tradiţionale.

Nivelul de resurse este eficient, dar are într-o măsură oarecare suport limitat pentru proprietăţile de conectare în cascadă, pentru programabilitatea evenimentelor şi a celei granulate fin. Proiectantul scenei are astfel abilitatea de a compara eficienţa cu programabilitatea, atât cât este nevoie.

În acord cu un aspect al prezentei invenţii, sistemul de grafică vectorială, la nivelul resurselor, se găseşte, de asemenea, în corelaţie cu stratul API al vizualului, în care marcatorul de nivel al resurselor, dintr-o implementare, este exprimat ca un "VisualBrush" (pensulă pentru vizual). Dacă marcatorul de resurse este analizat gramatical, este creat un obiect vizual. Obiectul vizual este setat într-un "VisualBrush" care poate fi utilizat de către conformaţii, de către comenzi, precum şi de alte elemente, la nivel de element.

Fig. 25 este o reprezentare a unei ierarhii **2500**, a clasei de elemente. Clasele modelului obiect al limbajului de marcare, din invenţia de faţă, sunt reprezentate prin căsuţe umbrite, şi includ o clasă **2502**, de conformaţii, o clasă **2504**, de imagini, o clasă video **2506** şi o clasă **2508**, de prelate. Elementele clasei de conformaţii includ dreptunghiul **2510**, polilinia **2512**, poligonul **2514**, traiectoria **2516**, linia **2518** şi elipsa **2520**. De remarcat că la unele implementări, un element cerc nu poate fi prezent ca fiind indicat de către căsuţa **2522**, cu linii întrerupte, din fig. 25, totuşi pentru scopurile diverselor exemple de aici, va fi descris elementul cerc **2522**. Fiecare element poate include sau poate fi asociat cu datele de umplere (proprietate), cu datele de mişcare a pensulei, cu datele de decupare, cu datele transformatei, cu datele efectului de filtru, precum şi cu datele de mascare.

Aşa după cum este descris mai jos, conformaţiile corespund la geometria care este desenată cu proprietăţi de prezentare moştenite şi conectate în cascadă. Proprietăţile de prezentare sunt folosite pentru a construi trăgătorul şi pensula necesare pentru a desena figurile. Într-o implementare, figurile sunt prezentatori completi, la fel ca şi alte elemente de comandă. Totuşi, la alte implementări, poate fi prevăzută o clasă **2508**, de prelate, ca un recipient pentru figuri, iar figurile pot fi desenate numai dacă se găsesc într-un element de acoperire. De exemplu, pentru a păstra uşor figurile, acestea nu au permisiunea de a avea ataşaţi prezentatori. În schimb, prelata are un prezentator ataşat şi desenează figurile. Elementele prelatei sunt descrise mai jos, cu mai multe detalii.

Aşa după cum a fost, de asemenea, descris mai jos, clasa de imagini este mai specifică decât o conformaţie, iar, de exemplu, poate include date de delimitare, care pot fi complexe. De exemplu, o bordură poate fi specificată ca o culoare în partea de sus, ca o culoare diferită pe părţile laterale, cu posibilitatea specificării diferitelor grosimi şi a setării altor proprietăţi. Poziţia, rotaţia formei şi scalarea pot fi setate pentru o imagine sau pentru un element similar, închis în căsuţă, cum ar fi textul sau imaginea video. De notat faptul că elementele de imagine şi cele video pot exista şi pot fi prezentate în afara unui element de acoperire, şi pot să succeadă şi din "BoxedElement" (element închis într-o căsuţă), de exemplu, pentru a obţine fundalul, bordurile, precum şi suportul de umplutură de la acest element.

Elementul video permite imaginii video (sau multimedia similară) de a fi redată în interiorul unui element afişat. În această manieră, sistemul de grafică vectorială asigură o interfață de marcare la stratul API, care este fără îndoială consistentă în cadrul multimedia, incluzând text, grafică 2D, grafică 3D, animație, video, imagini nemişcate, precum şi audio. Aceasta permite designerilor să învețe să lucreze cu o "media", pentru a integra mai uşor o altă media în aplicații şi documente. Sistemul de grafică vectorială dă, de asemenea, posibilitatea ca multimedia să fie animată în acelaşi mod ca şi alte elemente, acordă din nou designerilor abilitatea de a folosi multimedia, la fel ca alte elemente, fără a sacrifica încă unicitatea intrinsecă de memorie, a fiecărui tip individual din "media". De exemplu, un designer poate utiliza aceeaşi schemă de denumire pentru rotație, scalare, animație, desenare, compunere, precum şi alte efecte, dintr-o parte în cealaltă a diferitelor tipuri de "media", prin care designerii pot crea uşor aplicații foarte bogate, tot atât de bine permițând unei implementări foarte eficiente de remitere şi de compunere de a fi construită dedesubt.

Fig. 26 reprezintă o implementare la care codul de marcare **2602** este interpretat de către un analizor gramatical/traducător **2604**. În general, analizorul gramatical/traducătorul **2604** adaugă elemente la un arbore de elemente/sistem de proprietăți **208** (reprezentate, de asemenea, în fig. 2) şi ataşază prezentatori la aceste elemente. Sistemul prezentator **210** preia apoi arborele de elemente **210**, împreună cu prezentatorii ataşați, şi traduce datele în obiecte şi apelează stratul API al vizualului, **212**. De remarcat că nu toate elementele sunt necesar a fi traduse, ci numai acelea cu prezentatori ataşați.

În general, un element este un obiect din stratul de elemente care participă la sistemul de proprietăți, la sistemul de evenimente, precum şi la cel de amplasare/prezentare. Analizorul gramatical găseşte etichetele (tag-uri) şi decide dacă aceste etichete ajută la definirea unui element sau a unui obiect resursă. În cazul special al unui "VisualBrush", aceleaşi etichete pot fi interpretate drept elemente sau, de asemenea, pot fi interpretate ca obiecte resurse, depinzând de contextul unde apar aceste etichete, de exemplu, depinzând de faptul, dacă apar sau nu în sintaxa complexă de proprietăți.

În conformitate cu un aspect al prezentei invenții, limbajul de marcare asigură căi distincte de a descrie o resursă, ce include un singur format al şirului sau o notație complexă a obiectelor. Pentru un format simplu al şirului, analizorul gramatical/traducătorul **2604** utilizează un convertor de tipuri, **2608**, pentru conversia unei succesiuni într-un obiect API, specific al vizualului. Cu titlu de exemplu, la următoarea linie a marcatorului, valoarea proprietății de "Fill" (umplere) poate fi convertită la un obiect pensulă, cu ajutorul convertorului de tipuri, **2608**:

```
<Circle CenterX="10" CenterY="10" Radius="5" Fill="Red" />
```

Aşa după cum poate fi uşor apreciat, conversia unei astfel de linii, în linie a marcatorului bazat pe etichete, este făcută împreună cu succesiuni simple de parametri, este făcută direct la obiectul pensulă şi asigură o cale simplă pentru un proiectant al scenei de a adăuga o conformație, precum şi însuşirile acesteia, la o scenă.

Totuşi, există momente când atributul de umplere este prea complex, pentru a se insera într-o singură succesiune. Într-o astfel de situație, sintaxa complexă de proprietăți, care poate fi în linie în cadrul marcatorului, este folosită pentru a seta această proprietate. De exemplu,

următoarea sintaxă complexă de proprietăți umple un cerc mai degrabă cu un gradient decât cu o culoare uniformă, specificând culorile diferitelor capete ale gradientului (care pot fi în domeniul de la 0 la 1):

```
<Circle  CenterX="10"  CenterY="10"  Radius="5">
   <Circle. Fill>
      <LinearGradient>
         <GradientStop  Color="Red"  Offset="0"/>
         <GradientStop  Color="Blue"  Offset="0.33"/>
         <GradientStop  Color="Green"  Offset="0.66"/>
         <GradientStop  Color="Red"  Offset="1.0"/>
      </LinearGradient>
   </Circle. Fill>
</Circle>
```

În plus, la a fi prezent în linie în cadrul marcatorului, un exemplu, de resursă, poate fi localizat în altă parte (cum ar fi în marcator sau într-un fişier, care pot fi locali sau pe o rețea aflată la distanță şi care este descărcată în mod adecvat), şi poate fi referit printr-un nume (de exemplu, un nume de text, un identificator de referință sau un altul corespunzător). În acest mod, un designer de scenă poate refolosi un element din arborele de elemente, de la un capăt la celălalt al unei scene, incluzând elementele descrise de către sintaxa complexă de proprietăți.

Analizorul gramatical manevrează marcatorul din sintaxa complexă de proprietăți, prin accesarea, atât cât este necesar, a convertorului de tipuri, **2608**, şi, de asemenea, prin combinarea parametrilor specifici la proprietățile obiectului, prin aceasta manevrând complexitatea pentru designerul de scenă. Astfel, analizorul gramatical nu numai fixează obiectele, ci şi setează atributele asupra obiectelor. De notat că analizorul gramatical efectiv inițiază imediat un constructor să creeze obiectele, deoarece obiectele sunt invariabile.

Deoarece acelaşi model de remitere este împărțit între nivelul elementului şi nivelul API, multe dintre obiecte sunt în fond aceleaşi. Acest lucru face analiza gramaticală/traducerea mult mai eficientă şi permite, de asemenea, diferitelor tipuri de limbaje de programare (cum ar fi limbajele asemănătoare cu C#) abilitatea de a converti mai uşor dintr-o marcare în propria ei sintaxă şi viceversa. De remarcat faptul că, aşa cum este reprezentat în fig. 26, un alt astfel de limbaj de programare, **2610**, poate adăuga elemente la arborele de elemente **208** sau poate să se interfațeze direct cu stratul API al vizualului, **212**.

Aşa cum este reprezentat şi în fig. 26, precum şi în conformitate cu un aspect al invenției de față, acelaşi marcator **2602** poate fi utilizat pentru a programa la un nivel element şi la un nivel resursă. Aşa cum este descris mai sus, nivelul element dă proiectantului de decor o programabilitate completă, o utilizare a sistemului de proprietăți care asigură succesiunea (de exemplu, trăsăturile asemănătoare cu un aspect de coală de hârtie), precum şi o desfăşurare a evenimentelor (de exemplu, prin care un element poate avea ataşat un cod, pentru a-şi schimba aspectul, poziția, ş.a.m.d., ca răspuns la un eveniment de intrare al utilizatorului). Totuşi, prezenta invenție prevede, de asemenea, un mecanism la nivel de resursă, prin care designerii scenici pot în fond să scurtcircuiteze arborele de elemente şi sistemul de prezentatori, şi pot programa direct la stratul API al vizualului. Pentru multe tipuri de figuri statice, imagini şi altele asemănătoare, unde nu sunt necesare trăsături la nivel de element, acest lucru asigură

o cale mai eficientă şi mai uşoară, pentru a extrage obiectul potrivit. În acest scop, analizorul gramatical recunoaşte dacă este prezentă o umplere de tipul "pensulă pentru vizuale" şi apelează direct stratul API **212**, împreună cu datele **2612**, ale nivelului resursă, pentru a crea obiectul. Cu alte cuvinte, aşa după cum este reprezentat în fig. 22, grafica vectorială a nivelului element obţine analizarea gramaticală din elementele create, care au nevoie ulterioară de traducere în obiecte, în timp ce grafica vectorială a nivelului resursă obţine, într-un mod eficient, analizarea gramaticală şi stocarea directă.

Cu titlu de exemplu, următoarea marcare este direct derivată din modelul obiect pentru obiectul "LinearGradient" (gradient liniar) şi umple un cerc exterior cu un "VisualBrush". Conţinutul acestui "VisualBrush" este definit de către marcarea interioară. De notat faptul că această sintaxă este în mod uzual folosită, pentru a exprima diferite pensule, transformate şi animaţii:

```
<Circle CenterX="10" CenterY="10" Radius="5">
   <Circle. Fill>
      <VisualBrush xmlns="...">
         <Circle CenterX="0.5" CenterY="0.5" Radius="0.25" Fill="Blue"/>
         <Circle CenterX="0.6" CenterY="0.6" Radius="0.25" Fill="Green"/>
         <Circle CenterX="0.7" CenterY="0.7" Radius="0.25" Fill="Red"/>
         <Circle CenterX="0.8" CenterY="0.8" Radius="0.25"
          Fill="LemonChiffon"/>
      </VisualBrush>
   </Circle. Fill>
</Circle>
```

De notat că, în timp ce aceste obiecte umplute cu pensula de vizual sunt stocate în mod eficient, datele nivelului resursă (sau obiectele create prin aceasta) pot fi completate cu elemente şi cu o porţiune din arborele de elemente, **208**, aşa cum este reprezentat la modul general în fig. 26. În acest scop, aceste resurse ale pensulei de vizual pot fi denumite (de exemplu, cu un nume, o referire sau un alt identificator potrivit) şi completate la fel ca şi alte resurse descrise cu ajutorul sintaxei complexe de proprietăţi.

Întorcându-ne la o explicare a prelatei, aşa după cum este menţionat mai sus într-o implementare alternativă, conformaţiile pot fi păstrate cu uşurinţă, şi astfel se poate pretinde ca să fie conţinute într-o prelată. La această implementare alternativă, dacă conţinutul este remis, acesta este remis la o prelată infinită, independentă de dispozitiv, care are un sistem de coordonate asociat. Elementul de acoperire poate astfel poziţiona conţinutul, în conformitate cu coordonatele absolute. Elementul prelată poate, în mod opţional, să definească un "viewport" (port de vizualizare), care specifică decuparea, o transformată, un raport preferat al aspectelor şi o cale de cartografiere ("map-are") a "viewporf'-ului într-un domeniu "părinte". Dacă nu este stabilit niciun "viewport", elementul prelată poate specifica numai o grupare a primitivelor de desenare şi poate fixa o transformată, o opacitate, precum şi alte atribute de compunere.

Ceea ce urmează este un exemplu de marcator pentru o prelată mostră:

```
<Canvas Background="black" Top="100" Left="100" Height="600" Width="800">
   <Rectangle Top="600" Left="100" Width="100" Height="50" Fill="red"
Stroke="blue" StrokeWidth="10" />
  <Line x1="100" y1="300" x2="300" y2="100" Stroke="green" StrokeWidth="5"
/ >
</Canvas>
```

De notat că la o implementare, dacă coordonatele sunt specificate fără unități de măsură, atunci acestea sunt considerate drept "pixeli logici", având a 96-a parte dintr-un inch, iar în exemplul de mai sus, linia va avea lungimea de 200 pixeli. În afară de coordonate, alte proprietăți includ lățimea, alinierea pe orizontală şi pe verticală a înălțimii, precum şi "ViewBox"-ul (de tip "rect" rectangular; referința nu este fixată sau este (0, 0, 0, 0), însemnând că nu este făcută nicio ajustare, iar proprietățile de extindere şi de aliniere sunt ignorate). Aşa după cum este descris la modul general mai sus, cu referire la fig. 18...20, alte proprietăți includ extinderea, care, atunci când nu este specificată, păstrează mărimea originală, sau poate:

- să specifice o umplere la care raportul aspectelor nu este păstrat, iar conținutul este scalat, pentru a umple suprafețele de separație, stabilite de către partea de sus/ stânga/l ățime/ înălțime,
- să specifice uniforma, care scalează mărimea în mod uniform, până ce imaginea se potriveşte la suprafețele de separație stabilite de către partea de sus/stânga/lățime/înălțime, sau
- să specifice "UniformToFill", care scalează mărimea în mod uniform, pentru a umple suprafețele de separație, stabilite de către partea de sus/stânga/lățime/înălțime, precum şi "clip"-urile, atât cât este necesar.

Pentru a se găsi mai departe în corelație cu modelul obiect de nivel scăzut, proprietatea de transformare stabileşte un nou reper de coordonate pentru copiii elementului, în timp ce proprietatea de a decupa limitează regiunea la care conținutul poate fi desenat pe prelată, cu traiectoria prestabilită de decupare, definită ca o căsuță de delimitare. Proprietatea de "ZIndex" poate fi folosită pentru a specifica comanda de remitere pentru elementele prelatei, incluse unele în altele, din interiorul unui panou.

"Viewbox"-ul specifică un nou sistem de coordonate pentru conținut, de exemplu, prin redefinirea întinderii şi a originii "viewport"-ului. Extinderea ajută la specificarea felului cum acest conținut se poziționează în "viewport". Valoarea atributului lui "viewBox" este o listă de patru numere "fără unități de măsură", <min-x>, <min-y>, <width> (lățime) şi <height> (înălțime), de exemplu separate prin zone albe şi/sau prin virgulă, şi este de tipul "Rect" (rectangular). "Rect"-ul lui "Viewbox" specifică dreptunghiul din spațiul utilizatorului care se poziționează pe căsuța de delimitare. Acesta lucrează la fel cu inserarea unui "scaleX" şi a unui "scaleY". Proprietatea de extindere (în cazul în care opțiunea este alta de cât "none" (niciuna)) asigură comanda suplimentară, pentru a păstra raportul aspectelor din grafică. O transformare suplimentară este aplicată la descendenții dintr-un element dat, pentru a realiza efectul specificat.

În exemplul de mai sus, rezultatul efectiv al dreptunghiului din mostra de marcare de mai sus, la fiecare regulă de extindere, ar fi:

```
None - from (100, 600) to (200, 650)

Fill - from (100, 100) to (900, 700)

Uniform - from (100, ?) to (900, ?) - noua înălțime ("height") va fi
400, şi va fi centrată pe baza lui "HorizontalAlign" (aliniere orizontală) şi
al lui "VerticalAlign" (aliniere verticală).

UniformToFill - from ( ?, 100) to ( ?, 700) - Noua lățime ("width") este
1200, şi va fi din nou centrată pe baza lui "HorizontalAlign" şi
al lui "VerticalAlign".
```

Dacă există o transformată asupra prelatei, aceasta este în fond aplicată deasupra (de exemplu, la arbore) poziționării la "Viewbox". De notat faptul că această poziționare va extinde orice element dintr-o prelată, de exemplu, căsuțe, text şi aşa mai departe, nu numai conformațiile. În plus, este de remarcat faptul că, dacă este specificat un "viewbox", prelata nu se mai potriveşte cu conținutul ei, dar mai de grabă are o mărime specifică. Dacă "y-width" (lățimea pe y) şi "y-height" (înălțimea pe y) sunt de asemenea specificate, atunci proprietățile de extindere/aliniere sunt folosite pentru a intercala "viewbox"-ul în lățimea şi înălțimea specificate.

Elementele din modelul obiect pot să aibă, fiecare, aplicat un atribut de "Clip". La unele elemente, îndeosebi conformații, acesta este expus în mod direct, ca o proprietate comună de timp de rulare a limbajului, în timp ce la alții (de exemplu, majoritatea comenzilor), această proprietate este setată printr-o "DynamicProperty" (proprietate dinamică).

În general, calea de decupare restrânge regiunea la care poate fi desenat conținutul, aşa cum este reprezentat la modul general în fig. 27, în care este prezentat un buton dintr-o con figurație **2702**, nedecupată, precum şi dintr-o configurație **2704**, în care este specificată o cale de decupare (unde linia întreruptă reprezintă calea de decupare). În mod conceptual, oricare parte, dintr-un desen care se întinde în afara regiunii delimitate de calea activă, curentă de decupare, nu este desenată. O cale de decupare poate fi gândită ca o mască, în care acei pixeli din afara căii de decupare sunt negri, împreună cu o valoare "alpha" de zero, iar acei pixeli din interiorul căii de decupare sunt albi, cu o valoare "alpha" de unu (cu excepția posibilă de anti-altfel de denumire de-a lungul marginii siluetei).

O cale de decupare este definită de către un obiect din "Geometry" (geometrie), fie în linie, fie într-un mod caracteristic într-o secțiune a resursei. O cale de decupare este utilizată şi/sau completată, folosind proprietatea de "Clip" asupra unui element, aşa cum se arată în următorul exemplu:

```
<def: Resources>
      <Geometry  def: ID="MyClip">
            <Path  Data="..."  />
            <Rectangle  ...  />
         </Geometry>
</def: Resources>
      <Element  Clip=" "%resource;  MyClip"     ...  />
```

De notat faptul că animarea unui "Clip" este similară cu animarea transformatelor:

```
<Element>
      <Element.  Clip>
            <Circle  ... />
            <Rectangle  ...  >
                        <FloatAnimation  ...  />
            </Rectangle>
       </Element. Clip>
      ...  children  ...
</Element>
```

O cale este desenată prin specificarea datelor de "Geometry" şi a proprietăților de remitere, cum ar fi "Fill" (umplere), "Stroke" (mişcare a pensulei), precum şi "StrokeWidth" (lățimea de mişcare a pensulei) asupra elementului "Path" (cale). Un marcator exemplu pentru o cale este specificat după cum urmează:

```
<Path Data="M 100 100 L 300 100 L 200 300 z"
Fill="red" Stroke="blue" StrokeWidth="3" />
```

Şirul 'Data' al căii este de tip "Geometry". Un mod cu mai multe cuvinte şi mai complet de a specifica o cale desenată este prin sintaxa complexă de proprietăți, aşa cum este descrisă mai sus. Marcatorul (cum ar fi cel din următorul exemplu) este introdus direct în clasele de "Geometry" ale constructorului, descrise mai sus:

```
<Path>
   <Path. Data>
      <CircleGeometry ... />
      <RectangleGeometry ... />
      <PathGeometry ... />
   </Path. Data>
   <Path. Fill value="red" />
   <Path. Stroke value="blue" />
</Path>
```

Şirul de date al căii este de asemenea descris, utilizând următoarea notație pentru a descrie gramatica pentru un şir de date al traiectoriei:

```
*: 0 sau mai mult
+: 1 sau mai mult
?: 0 sau 1
( ): grupare
|: separă alternativele
duble ghilimele în jurul literelelor
```

Cele ce urmează arată informațiile din şirul de date al căii, descrise cu această notație (de notat că la o implementare, "FillMode" poate fi specificat aici, în locul unei proprietăți la nivelul de elemente):

```
wvg-path:
      wsp* moveto-drawto-command-groups? wsp*

moveto-drawto-command-groups:
      moveto-drawto-command-group
      | moveto-drawto-command-group wsp* moveto-drawto-command-
groups
```

```
moveto-drawto-command-group:
      moveto wsp* drawto-commands?

drawto-commands:
      drawto-command
      | drawto-command wsp* drawto-commands

drawto-command:
      closepath
      | lineto
      | horizontal-lineto
      | vertical-lineto
      | curveto
      | smooth-curveto
      | quadratic-bezier-curveto
      | smooth-quadratic-bezier-curveto
      | elliptical-arc

moveto:
      ( "M" | "m" ) wsp* moveto-argument-sequence

moveto-argument-sequence:
      coordinate-pair
      | coordinate-pair comma-wsp? lineto-argument-sequence

closepath:
      ("Z" | "z")

lineto:
      ( "L" | "l" ) wsp* lineto-argument-sequence

lineto-argument-sequence:
      coordinate-pair
      | coordinate-pair comma-wsp? lineto-argument-sequence

horizontal-lineto:
      ( "H" | "h" ) wsp* horizontal-lineto-argument-sequence

horizontal-lineto-argument-sequence:
      coordinate
      | coordinate comma-wsp? horizontal-lineto-argument-
sequence

vertical-lineto:
      ( "V" | "v" ) wsp* vertical-lineto-argument-sequence

vertical-lineto-argument-sequence:
      coordinate
```

```
        | coordinate comma-wsp? vertical-lineto-argument-sequence

curveto:
        ( "C" | "c" ) wsp* curveto-argument-sequence

curveto-argument-sequence:
        curveto-argument
        | curveto-argument comma-wsp? curveto-argument-sequence

curveto-argument:
        coordinate-pair comma-wsp? coordinate-pair comma-wsp?
coordinate-pair

smooth-curveto:
        ( "S" | "s" ) wsp* smooth-curveto-argument-sequence

smooth-curveto-argument-sequence:
        smooth-curveto-argument
        | smooth-curveto-argument comma-wsp? smooth-curveto-
argument-sequence

smooth-curveto-argument:
        coordinate-pair comma-wsp? coordinate-pair

quadratic-bezier-curveto:
        ( "Q" | "q" ) wsp* quadratic-bezier-curveto-argument-
sequence

quadratic-bezier-curveto-argument-sequence:
        quadratic-bezier-curveto-argument
        | quadratic-bezier-curveto-argument comma-wsp?
            quadratic-bezier-curveto-argument-sequence

quadratic-bezier-curveto-argument:
        coordinate-pair comma-wsp? coordinate-pair

smooth-quadratic-bezier-curveto:
        ( "T" | "t" ) wsp* smooth-quadratic-bezier-curveto-
argument-sequence

smooth-quadratic-bezier-curveto-argument-sequence:
        coordinate-pair
        | coordinate-pair comma-wsp? smooth-quadratic-bezier-
curveto-argument-sequence

elliptical-arc:
        ( "A" | "a" ) wsp* elliptical-arc-argument-sequence
```

```
elliptical-arc-argument-sequence:
        elliptical-arc-argument
        | elliptical-arc-argument comma-wsp? elliptical-arc-
argument-sequence

elliptical-arc-argument:
        nonnegative-number comma-wsp? nonnegative-number comma-
wsp?
                number comma-wsp flag  comma-wsp flag comma-wsp
coordinate-pair

coordinate-pair:
        coordinate comma-wsp? coordinate

coordinate:
        number

nonnegative-number:
        integer-constant
        | floating-point-constant

number:
        sign? Integer-constant
        | sign? floating-point-constant

flag:
        "0" | "1"

comma-wsp:
        (wsp+ comma? wsp*) | (comma wsp*)

comma:
        ","

integer-constant:
        digit-sequence

floating-point-constant:
        fractional-constant exponent?
        | digit-sequence exponent

fractional-constant:
        digit-sequence? "." digit-sequence
        | digit-sequence "."

exponent:
        ( "e" | "E" ) sign? digit-sequence
```

```
sign:
        "+" | "-"
digit-sequence:
        digit
        | digit digit-sequence

digit:
        "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"

wsp:
        (#x20 | #x9 | #xD | #xA)
```

Elementul imagine (fig. 25) indică faptul că respectivul conținut al unui fişier complet este pe cale a fi remis într-un dreptunghi dat, din interiorul sistemului de coordonate curent al utilizatorului. Imaginea (indicată prin semnul de imagine) se poate referi la fişierele rastru de imagine, cum ar fi PNG sau JPEG, sau la fişiere cu tipul MIME din "image/wvg", aşa cum se începe în următorul exemplu:

```
<Image Top="200" Left="200" Width="100px" Height="100px"
Source="myimage. png">
</Image>
```

Următorul tabel asigură informațiile pentru imagini, prin câteva proprietăți exemple:

Tabel

Denumire	**Tip**	**R/RW**	**Valoare standard**	**Descriere**
Top	BoxUnit			Coordonate pentru partea de sus a lui "Image"(imagine)
Left	BoxUnit			Coordonate pentru partea stângă a lui "Image"
Width	BoxUnit			Lățimea lui "Image"
Height	BoxUnit			Înălțimea lui "Image"
Source	ImageData			Sursa lui "Image"
Dpi	Float		96 (?)	Ținta DPI pentru utilizarea la dimensionare
HorizontalAlign	enum { Left (?), Center (?), Right (?) }		Center	
VerticalAlign	enum { Top (?), Middle (?), Bottom (?) }		Middle	

Tabel (continuare)

Denumire	Tip	R/RW	Valoare standard	Descriere
Stretch	enum Stretch { None, Fill, Uniform, UniformToFill, }		None	**None:** Păstrează forma inițială **Fill:** Raportul aspectelor nu este păstrat, iar conținutul este scalat, pentru a umple bordurile stabilite de tlbh **Uniform:** Forma de scalare uniformă, până ce imaginea se potriveşte cu bordurile stabilite prin tlwh. **UniformToFill:** Forma de scalare pentru a umple uniform bordurile stabilite prin tlbh, şi decupate
ReadyState	enum { MetaDataReady, Loading, Loaded LoadError }			
LoadCounter	Int	Read	Null	Numărătorul care se incrementează atunci când "ReadyState" se încarcă ("Loading")
Name	String			Alternează text pentru "Image"

Aşa cum este descris mai sus, conformațiile corespund geometriei desenate cu ajutorul proprietăților de prezentare moştenite şi prezentate în cascadă. Următoarele tabele pornesc de la proprietățile exemplelor de figuri, pentru elementele de bază ale figurilor descrise mai sus ("Rectangle" - dreptunghi, "Ellipse" - elipsă, "Line" - linie, "Polyline" - polilinie, "Polygon" - poligon). De remarcat faptul că aceste figuri de bază pot avea proprietăți de mişcare a pensulei, proprietăți de umplere, şi pot fi utilizate drept căi de decupare, pot avea caracteristici de succesiune şi se aplică la ambele niveluri de element şi de "Resource" (resursă):

Tabel

Denumire	Tip	R/RW	Valoare standard	Descriere
Fill	Brush	RW	null	Coordonate pentru partea de sus a lui "rect"
FillOpacity	Float	RW	1.0	Coordonate pentru partea stângă a lui "rect"
Stroke	Brush	RW	null	Lățimea lui "rect"
StrokeOpacity	Float	RW	1.0	Înălțimea lui "rect"
StrokeWidth	BoxUnit	RW	lpx	Lățimea mişcării pensulei. lpx = 1/96 dintr-un inch
FillRule	enum { EvenOdd, NonZero, }	RW	EvenOdd	FillRule indică algoritmul care este de utilizat pentru a determina care părți din prelată sunt incluse în interiorul figurii.

Tabel (continuare)

Denumire	Tip	R/RW	Valoare standard	Descriere
StrokeLineCap	enum { Butt, Round, Square, Diamond }	RW	Butt	"StrokeLineCap" specifică figura ce trebuie utilizată la sfârşitul deschiderii sub-căilor, atunci când acestea sunt mişcate cu pensula.
StrokeLineJoint	enum { Miter, Round, Bevel }	RW	Miter	"StrokeLineJoint" specifică figura ce este utilizată la colţurile căilor (sau alte conformaţii vectoriale) care sunt mişcate cu pensula, atunci când acestea sunt mişcate cu pensula.
StrokeMiterLimit	Float	RW	4.0	Limita la raportul dintre "MiterLength" şi "StrokeWidth". Valoarea să fie >=1
StrokeDashArray	PointList	RW	null	"StrokeDashArray" comandă secvenţa de linii întrerupte şi de pauze utilizate la căile de mişcare a pensulei. <dasharray> conţine o listă de <numere> (<number>s) separate prin spaţii şi prin virgulă, care specifică lungimile liniilor punctate şi ale pauzelor alternative din unităţile utilizatorului. Dacă este asigurat un număr impar de valori, atunci lista de valori este repetată, pentru a produce un număr par de valori. Astfel, reţeaua de linii punctate de mişcare a pensulei: 5 3 2 este echivalentă cu reţeaua de linii punctate de mişcare a pensulei: 5 3 2 5 3 2 .
StrokeDashOff set	Point	RW		"StrokeDashOffset" specifică distanţa din secvenţa de linii punctate, la startarea liniilor punctate.
Transform	Transform	RW	null	Transformata stabileşte un nou reper de coordonate pentru copiii elementului
Clip	Geometry	RW	null	"Clip"-ul limitează regiunea la care vopseaua poate fi aplicată pe prelată. Calea standard de decupare este definită ca o căsuţă de delimitare.

Ceea ce urmează este o sintaxă exemplu de marcare pentru un dreptunghi:

```
<Rectangle Top="600" Left="100" Width="100" Height="50"
 Fill="red" Stroke="blue" StrokeWidth="10" />
```

Un dreptunghi are următoarele proprietăți în modelul obiect (de notat că dreptunghiurile sunt citite/scrise, au valori prestabilite egale cu zero, suportă succesiunea şi se aplică la ambele niveluri de elemente şi de resurse):

Tabel

Denumire	Tip	Descriere
Top	BoxUnit	Coordonate pentru partea de sus a lui "rect"
Left	BoxUnit	Coordonate pentru partea stângă a lui "rect"
Width	BoxUnit	Lățimea lui "rect"
Height	BoxUnit	Înălțimea lui "rect"
RadiusX	BoxUnit	Pentru dreptunghiuri rotunjite, raza pe axa X a elipsei, utilizată pentru a rotunji colțurile dreptunghiului. Dacă este specificată o rază pe axa X negativă, va fi utilizată valoarea absolută a razei.
RadiusY	BoxUnit	Pentru dreptunghiuri rotunjite, raza pe axa Y a elipsei, utilizată pentru a rotunji colțurile dreptunghiului. Dacă este specificată o rază pe axa Y negativă, va fi utilizată valoarea absolută a razei.

Ceea ce urmează este o sintaxă exemplu de marcare pentru un cerc:

```
<Circle CenterX="600" CenterY="100" Fill="red"
 Stroke="blue" StrokeWidth="10" />
```

Un cerc are următoarele proprietăți în modelul obiect (de notat că cercurile sunt citite/scrise, au valori prestabilite egale cu zero, suportă succesiunea şi se aplică la ambele niveluri de elemente şi de resurse):

Tabel

Denumire	Tip	Descriere
CenterX	BoxUnit	Coordonata X a centrului cercului
CenterY	BoxUnit	Coordonata Y a centrului cercului
Radius	BoxUnit	Raza cercului

Ceea ce urmează este o sintaxă exemplu de marcare pentru o elipsă:

```
<Ellipse CenterX="600" CenterY="100" Fill="red"
 Stroke="blue" StrokeWidth="10" />
```

O elipsă are următoarele proprietăți în modelul obiect (de notat că elipsele sunt citite/scrise, au valori prestabilite egale cu zero, suportă succesiunea şi se aplică la ambele niveluri de elemente şi de resurse):

Tabel

Denumire	Tip	Descriere
CenterX	Coordonată	Coordonata X a centrului elipsei
CenterY	Coordonată	Coordonata Y a centrului elipsei
RadiusX	Lungime	Raza pe axa X a elipsei. Dacă este specificată o rază pe axa X negativă, va fi utilizată valoarea absolută a razei.
RadiusY	Lungime	Raza pe axa Y a elipsei. Dacă este specificată o rază pe axa Y negativă, va fi utilizată valoarea absolută a razei.

Ceea ce urmează este o sintaxă exemplu de marcare pentru o linie:

```
<Line x1="100" y1="300" x2="300" y2="100"
 StrokeWidth="5" />
```

O linie are următoarele proprietăți în modelul obiect (de notat că liniile sunt citite/scrise, au valori prestabilite egale cu zero, suportă succesiunea şi se aplică la ambele niveluri de elemente şi de resurse):

Tabel

Denumire	Tip	Descriere
X1	BoxUnit	Coordonata pe axa X a începutului liniei. Valoarea prestabilită este "0".
Y1	BoxUnit	Coordonata pe axa Y a începutului liniei. Valoarea prestabilită este "0".
X2	BoxUnit	Coordonata pe axa X a sfârşitului liniei. Valoarea prestabilită este "0".
Y2	BoxUnit	Coordonata pe axa Y a sfârşitului liniei. Valoarea prestabilită este "0".

'Polyline' ("Polilinia") defineşte un set de segmente de linii drepte, conectate între ele. În mod caracteristic, o "polilinie" defineşte o conformație deschisă.

Ceea ce urmează este o sintaxă exemplu de marcare pentru o polilinie:

```
<Polyline Fill="None" Stroke="Blue" StrokeWidth="10 cm"
Points="50,375
150,375 150,325 250,325 250,375
350,375 350,250 450,250 450,375
550,375 550,175 650,175 650,375
750,375 750,100 850,100 850,375
950,375 950,25 1050,25 1050,375
1150,375" />
```

O polilinie are următoarele proprietăți în modelul obiect (de notat că liniile sunt citite/scrise, au valori prestabilite egale cu zero, suportă succesiunea şi se aplică la ambele niveluri de elemente şi de resurse):

Tabel

Denumire	Tip	Descriere
Points	PointCollection	Punctele care alcătuiesc polilinia. Valorile coordonatelor sunt în sistemul de coordonate al utilizatorului.

Elementul "Polygon" (poligon) defineşte o conformație închisă, ce cuprinde un set de segmente de linii drepte, conectate între ele. Ceea ce urmează este o sintaxă exemplu de marcare pentru un poligon:

```
<Polygon  Fill="red" Stroke="blue"  StrokeWidth="10"
      points="350,75  379,161  469,161  397,215
      423,301  350,250  277,301  303,215
      231,161  321,161"  />
```

Un poligon are următoarele proprietăți în modelul obiect (de notat că liniile sunt citite/scrise, au valori prestabilite egale cu zero, suportă succesiunea şi se aplică la ambele niveluri de elemente şi de resurse):

Tabel

Denumire	Tip	Descriere
Points	PointCollection	Punctele care alcătuiesc poligonul. Valorile coordonatelor sunt în sistemul de coordonate al utilizatorului. Dacă este prevăzut un număr impar de coordonate, atunci elementul este în eroare.

Gramatica pentru specificațiile punctelor din elementele de "polilinie" şi de "poligon" este descrisă, împreună cu următoarea notație:

```
*:  0  sau  mai mult
+:  1  sau  mai mult
?:  0 sau 1
( ):  grupare
|:   separă alternativele
duble ghilimele în jurul literalelor
```

Ceea ce urmează descrie specificațiile punctelor din elementele de 'Polyline' ("polilinie") şi de 'Polygon' ("poligon"), utilizând notația de mai sus:

```
list-of-points:
        wsp*  coordinate-pairs?  wsp*

coordinate-pairs:
        coordinate-pair
        | coordinate-pair  comma-wsp  coordinate-pairs

coordinate-pair:
        coordinate  comma-wsp  coordinate

coordinate:
        number

number:
        sign?  Integer-constant
        | sign?  floating-point-constant

comma-wsp:
        (wsp+  comma?  wsp*)  |  (comma  wsp*)

comma:
        ","

integer-constant:
        digit-sequence

floating-point-constant:
        fractional-constant  exponent?
        | digit-sequence  exponent

fractional-constant:
        digit-sequence?  "."  digit-sequence
        | digit-sequence  "."

exponent:
        (  "e"  |  "E"  )  sign?  digit-sequence

sign:
        "+"  |  "-"
digit-sequence:
         digit
         | digit  digit-sequence

digit:
         "0"  |  "1"  |  "2"  |  "3"  |  "4"  |  "5"  |  "6"  |  "7"  |  "8"  |  "9"

wsp:
         (#x20  |  #x9  |  #xD  |  #xA)+
```

Aşa după cum poate fi văzut din descrierea anterioară, detaliată, sunt prevăzute un sistem, o metodă şi un model element/obiect, care asigură, codului de program, mecanisme diverse, pentru a se interfața cu un graf scenic. Sistemul, metoda, precum şi modelul obiect sunt utilizate în mod direct, sunt încă puternice, flexibile şi extensibile.

În timp ce invenția este susceptibilă de diverse modificări şi de construcții alternative, anumite forme de realizare ilustrate de aceasta sunt prezentate în desene şi au fost descrise mai sus în detaliu. Ar trebui totuşi să se înțeleagă că nu există nicio intenție de a limita invenția la formele specifice prezentate, ci din contră, intenția este de a acoperi toate modificările, construcțiile alternative, precum şi altele echivalente, intrând în spiritul şi în scopul invenției.

Revendicări

1. Sistem pentru procesarea informațiilor grafice şi a altor informații video, pentru afişarea pe sisteme de calculatoare, **caracterizat prin aceea că** sistemul cuprinde:

- un motor de compunere şi de animație de nivel superior (**214**) şi un motor de nivel inferior (**218**), motorul de nivel superior fiind instanțiat pe o bază per-aplicație, iar motorul de nivel inferior deservind cereri de la mai multe aplicații;
- un limbaj marcator, limbajul marcator cuprinzând instrucțiuni grafice, instrucțiunile grafice cuprinzând un format şir şi o notație de obiect, notația de obiect cuprinzând elemente grafice de la o clasă de elemente grafice;
- un model obiect grafic cuprinzând:

a. un obiect vizual de clasă de bază (**500**), care este un container pentru conținut grafic, care asigură funcționalitatea de bază pentru alte tipuri vizuale şi din care derivă alte tipuri vizuale,

b. un obiect vizual de tip container (**501**), care este un container pentru elemente vizuale şi care poate conține alte obiecte vizuale de tip container,

c. un obiect vizual de tip desen (**502**), care este un container pentru conținut grafic, şi

d. o clasă de elemente grafice (**2500**), clasa de elemente cuprinzând o clasă de formă (**2502**), o clasă de imagine (**2504**), o clasă video (**2506**) şi o clasă Canavas (**2508**), iar clasa de elemente fiind integrată cu un sistem de proprietăți generale;

- un convertor de tip (**2608**), convertorul de tip fiind configurat să convertească o instrucțiune grafică în format şir într-un obiect de interfață pentru programare de aplicație (API) vizual;
- un parser/translator (**2604**), parserul/translatorul fiind configurat să:

a. interpreteze instrucțiunile grafice, instrucțiunile grafice cuprinzând apeluri de cod directe, apeluri de cod de model obiect şi instrucțiuni grafice scrise utilizând limbajul marcator,

b. acceseze convertorul de tip, convertorul de tip fiind configurat să convertească o instrucțiune grafică în format şir într-un obiect API vizual, şi

c. interpreteze codul marcator şi, la interpretarea codului marcator, să adauge elemente din clasa elementelor grafice la un element arbore;

- un sistem de prezentare (**210**), sistemul de prezentare fiind configurat să traducă arborii de elemente grafice în apeluri la un API vizual; un API vizual (**212**), API-ul vizual fiind configurat să:

a. interfațeze cu sistemul de prezentare, să interfațeze cu parserul-translatorul şi să interfațeze cu apelurile de cod directe de la limbajele de programare, şi

b. ca răspuns la cererile sistemului de prezentare, parserul-translatorul creează obiecte de scenă în cadrul unui graf scenic; şi

- o interfață de afişare operabilă să faciliteze afişarea obiectelor grafice în cadrul grafului scenic.

2. Sistem conform revendicării 1, **caracterizat prin aceea că** elementele din modelul obiect de elemente sunt corelate cu obiectele din modelul obiect al grafului scenic.

3. Sistem conform revendicării 1, **caracterizat prin aceea că** marcatorul include textul în linie, ce include un şir care defineşte o proprietate a elementului, iar translatorul comunică cu un convertor de tip, pentru a converti şirul într-o proprietate a obiectului.

4. Sistem conform revendicării 1, **caracterizat prin aceea că** marcatorul include textul în linie, ce conține sintaxa de proprietate, sintaxa de proprietate specificând atribute multiple ale obiectelor grafice vectoriale.

5. Sistem conform revendicării 4, **caracterizat prin aceea că** textul în linie este identificat cu o referință care se referă la o altă locație din marcator.

6. Sistem conform revendicării 4, **caracterizat prin aceea că** textul în linie este identificat cu o referință care se referă la un fişier.

7. Sistem conform revendicării 4, **caracterizat prin aceea că** textul în linie este identificat cu o referință care corespunde la un fişier care poate fi descărcat dintr-o locație aflată la distanță, într-o rețea.

8. Sistem conform revendicării 1, **caracterizat prin aceea că** marcatorul include textul în linie, ce cuprinde sintaxa de proprietate complexă corespunzând unei resurse grafice.

9. Sistem conform revendicării 8, **caracterizat prin aceea că** resursa grafică descrie un obiect pensulă vizual, parserul/traducătorul asigurând datele nivelului resursă, pentru comunicarea directă cu stratul API vizual, pentru a crea un obiect vopsea vizual, ce corespunde elementului descris de către sintaxa de proprietate complexă.

10. Sistem conform revendicării 9, **caracterizat prin aceea că** datele nivelului resursă sunt identificate cu o referință care se referă la o altă locație din marcator.

11. Sistem conform revendicării 9, **caracterizat prin aceea că** datele nivelului resursă sunt identificate cu o referință care se referă la un fişier.

12. Sistem conform revendicării 9, **caracterizat prin aceea că** datele nivelului resursă sunt identificate cu o referință care se referă la un fişier care poate fi descărcat dintr-o locație aflată la distanță, într-o rețea.

13. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele modelului obiect grafic cuprinde un element imagine.

14. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele grafice cuprinde un element polilinie.

15. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele grafice cuprinde un element poligon.

16. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele grafice cuprinde un element traiectorie.

17. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele grafice cuprinde un element linie.

18. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele grafice cuprinde un element elipsă.

19. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele grafice cuprinde un element cerc.

20. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele grafice include date despre proprietatea de umplere.

21. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele grafice include date despre proprietatea de mişcare a pensulei.

22. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele grafice include date despre proprietatea de decupare.

23. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele grafice include date despre proprietatea de transformare.

24. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele grafice include date despre efect.

25. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele grafice include date despre opacitate.

26. Sistem conform revendicării 1, **caracterizat prin aceea că** unul dintre elementele grafice include date despre modul de îmbinare.

27. Sistem conform revendicării 1, **caracterizat prin aceea că** translatorul solicită instanțierea a cel puțin unui constructor, pentru a crea obiectele.

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

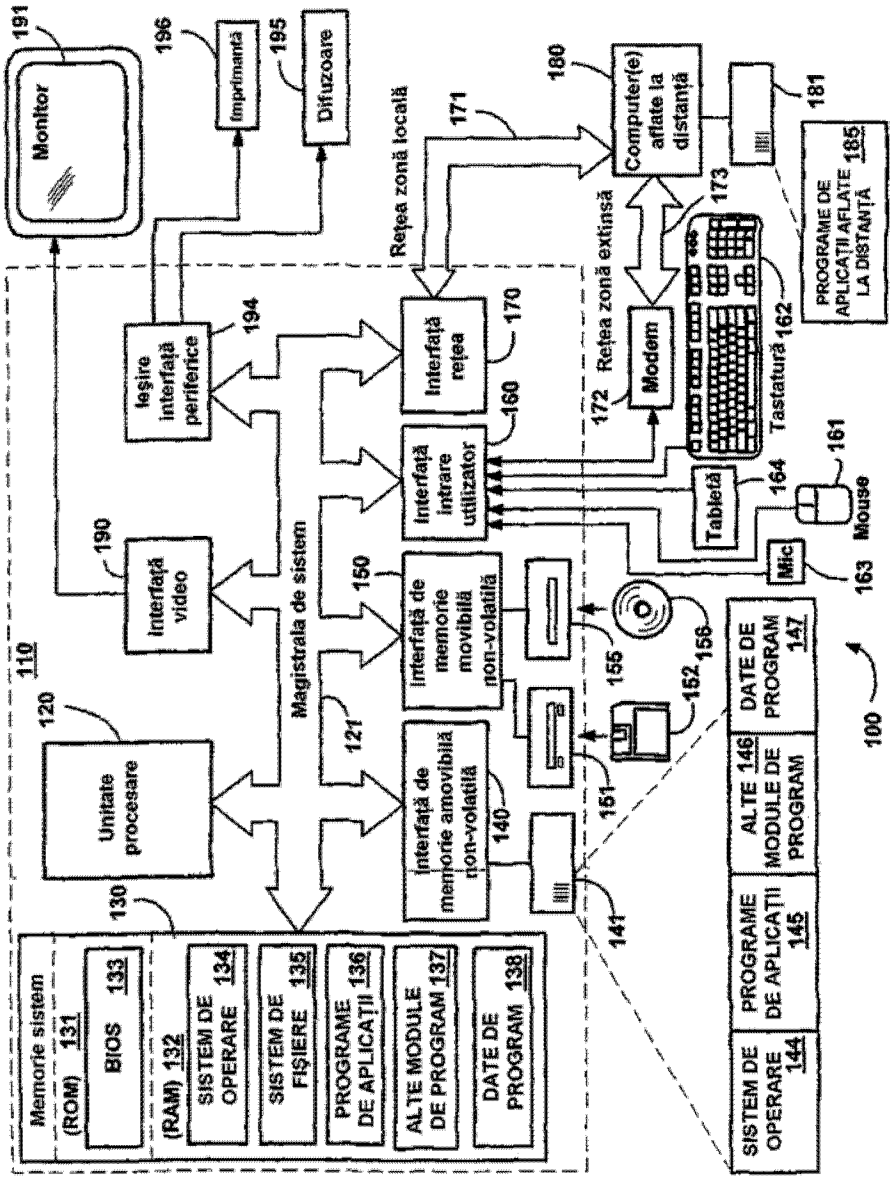

Fig. 1

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

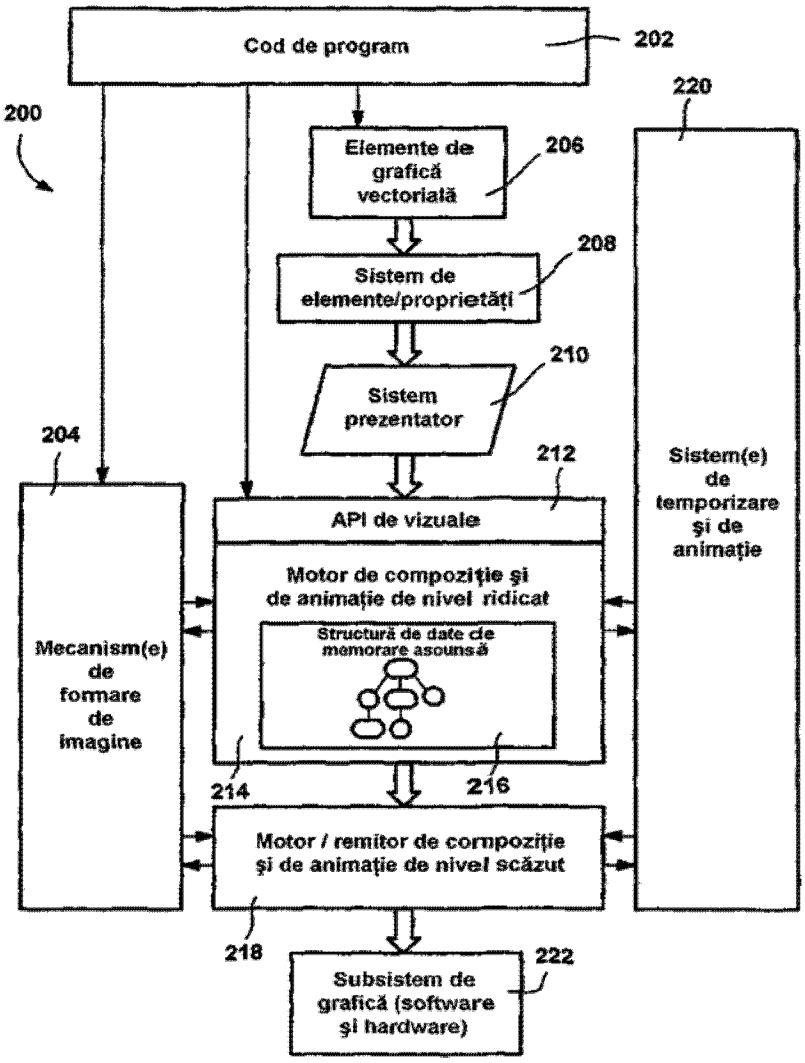

Fig. 2

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

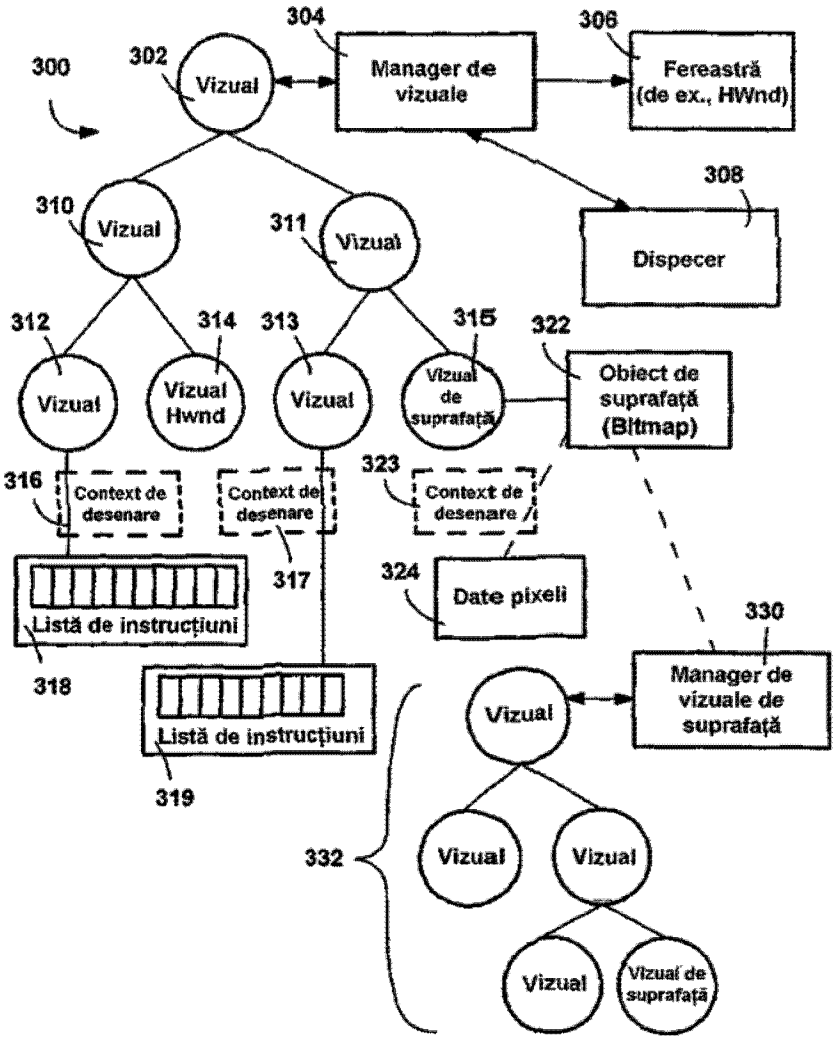

Fig. 3

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

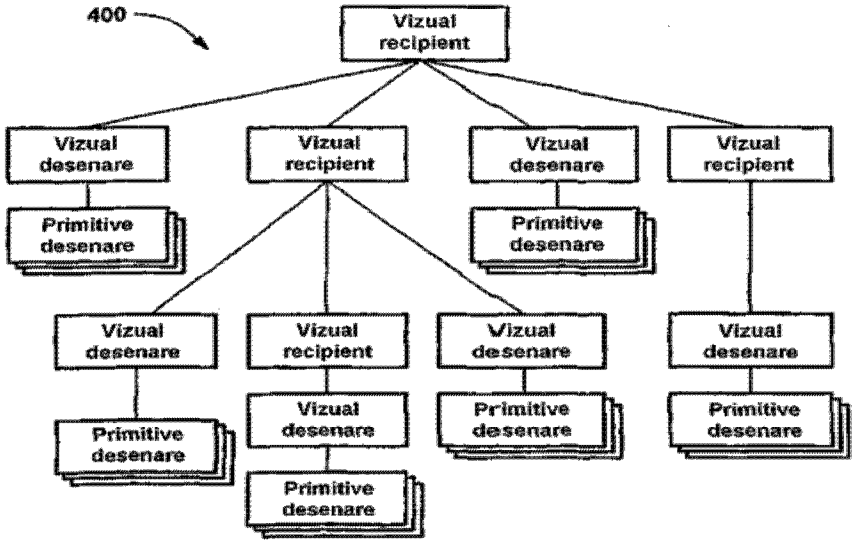

Fig. 4

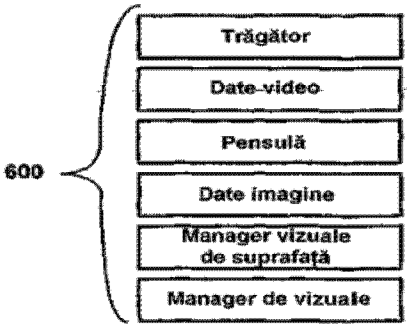

Fig. 6

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

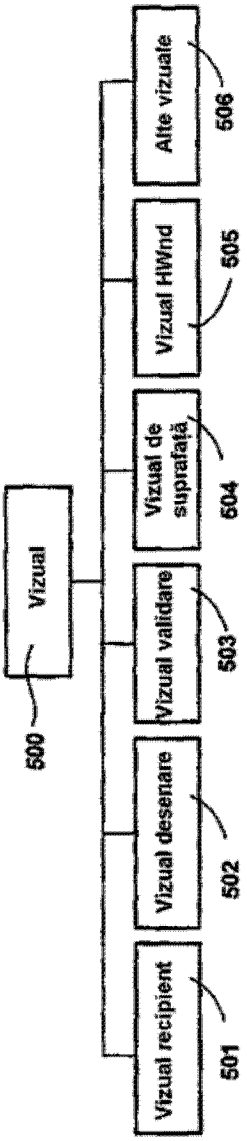

Fig. 5

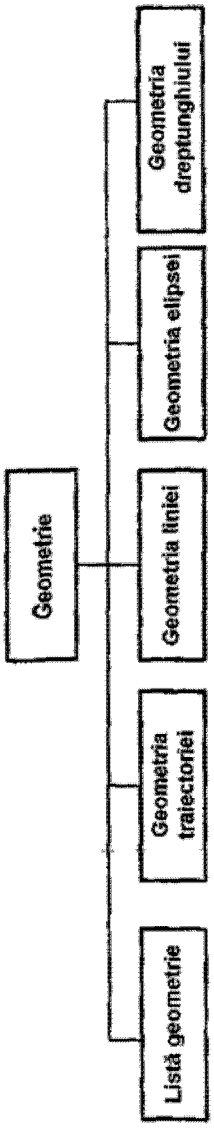

Fig. 12

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

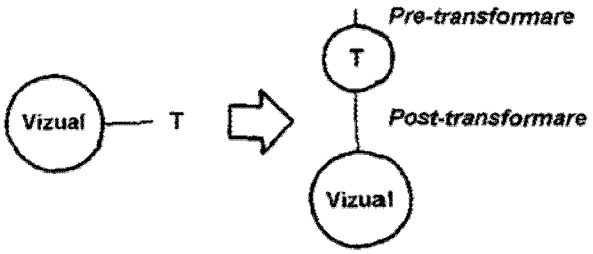

Fig. 7

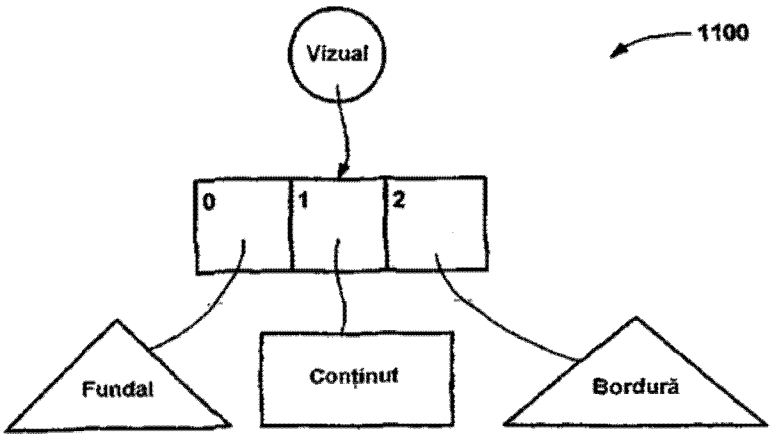

Fig. 11

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

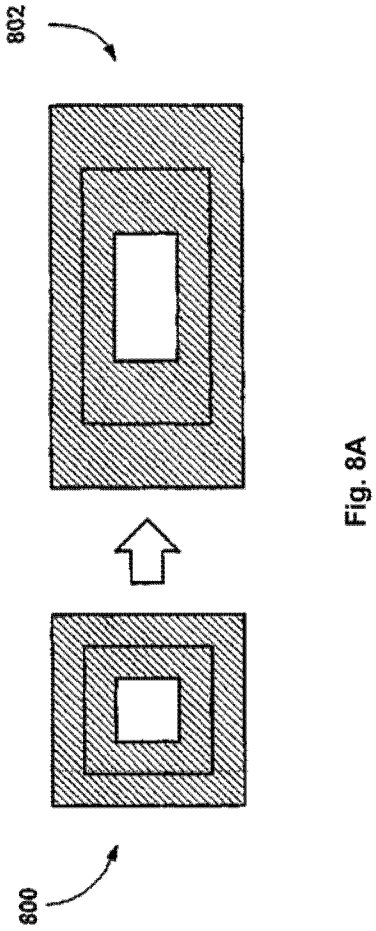

Fig. 8A

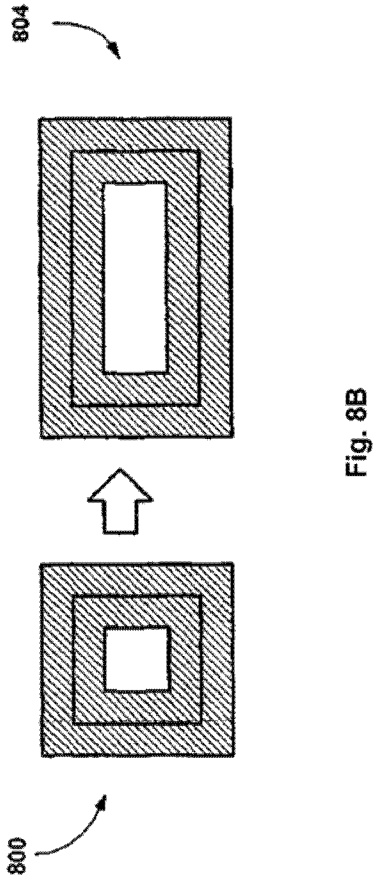

Fig. 8B

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

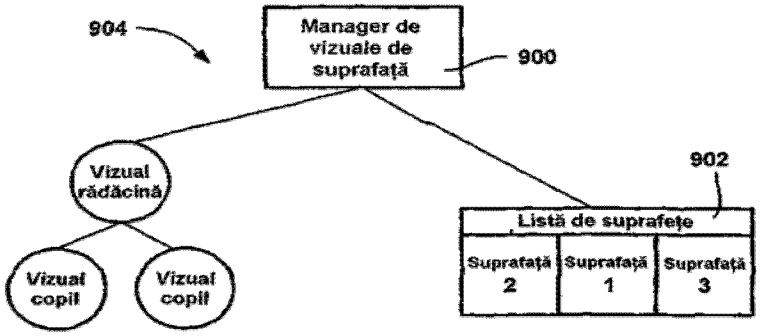

Fig. 9A

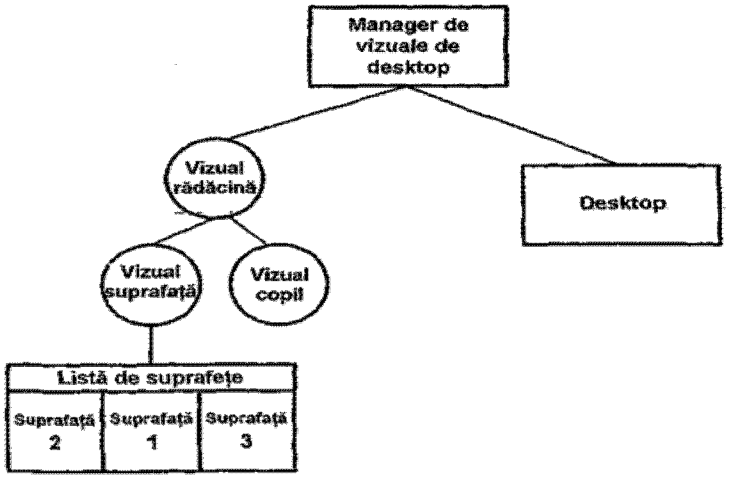

Fig. 9B

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

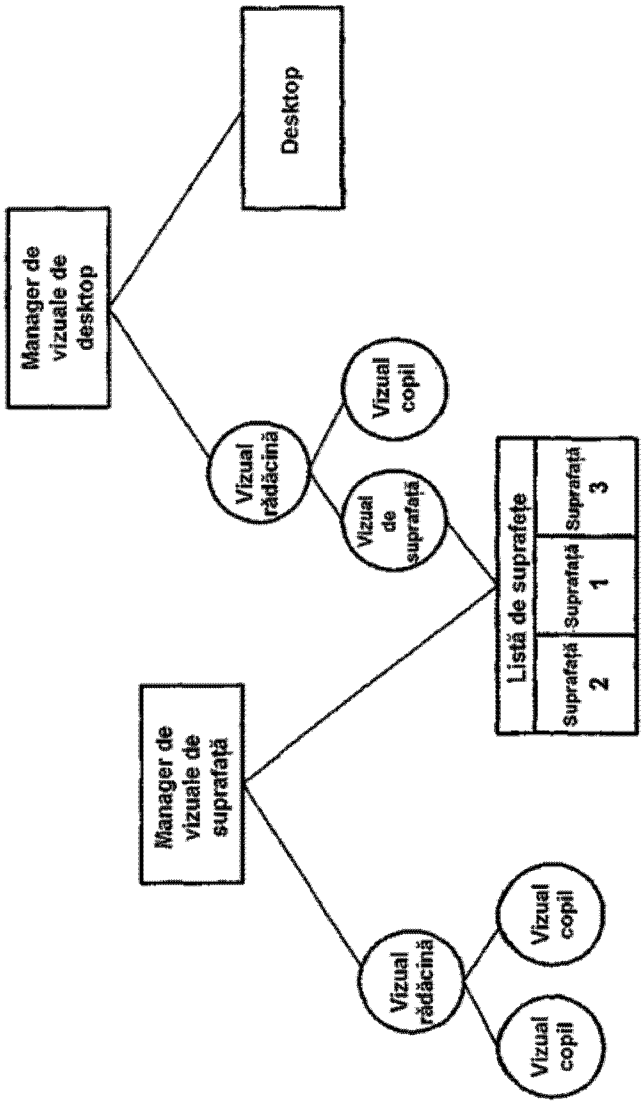

Fig. 9C

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

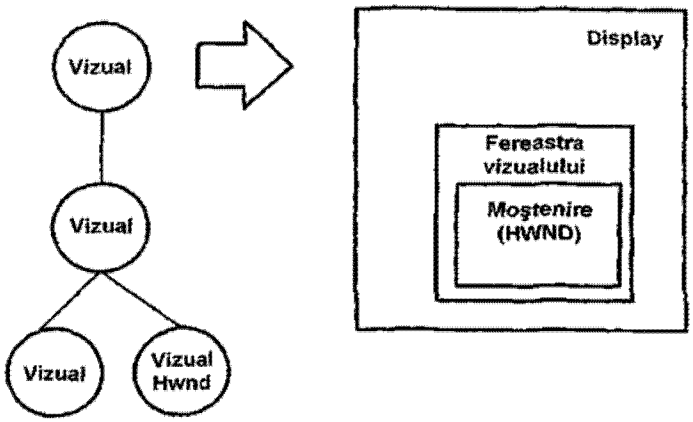

Fig. 10A

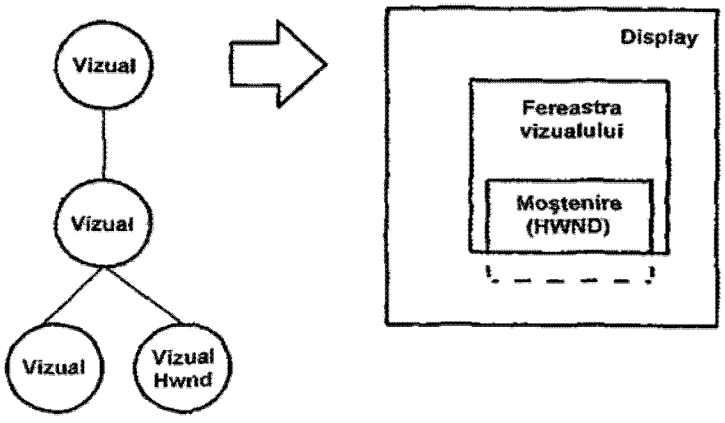

Fig. 10B

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

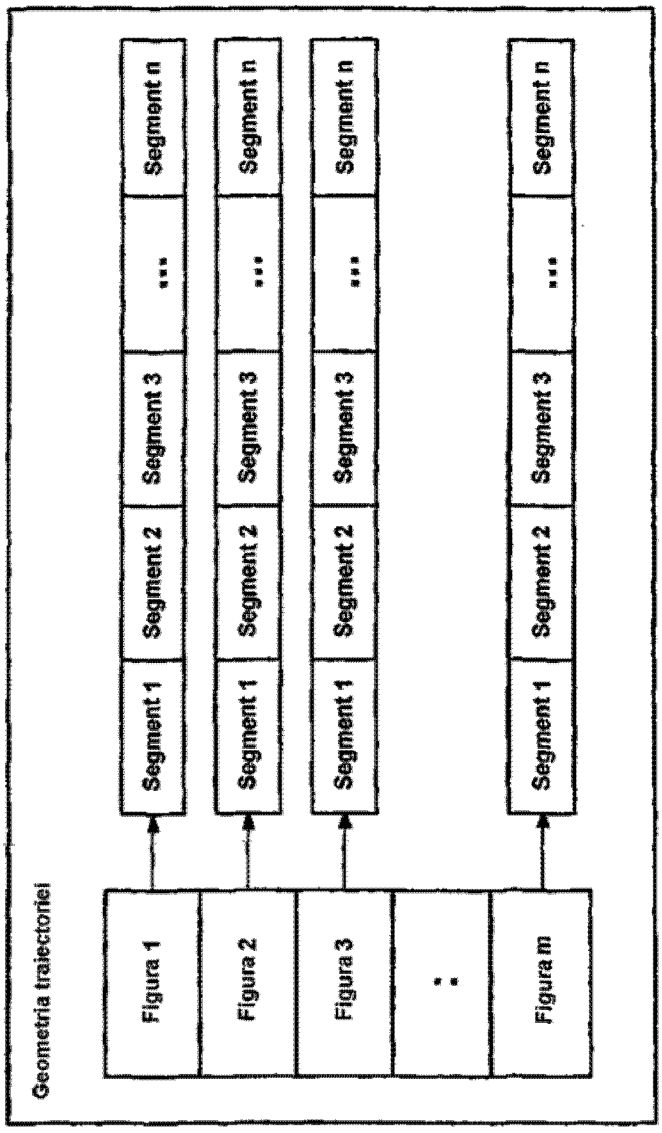

Fig. 13

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

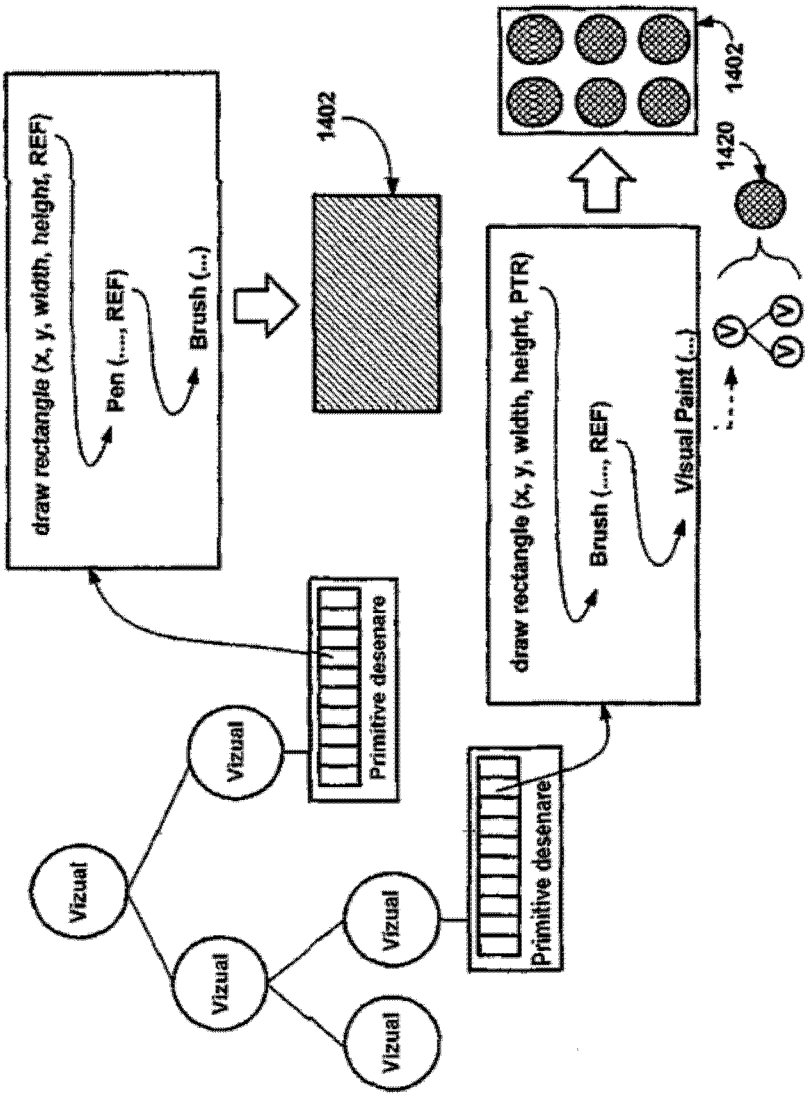

Fig. 14

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

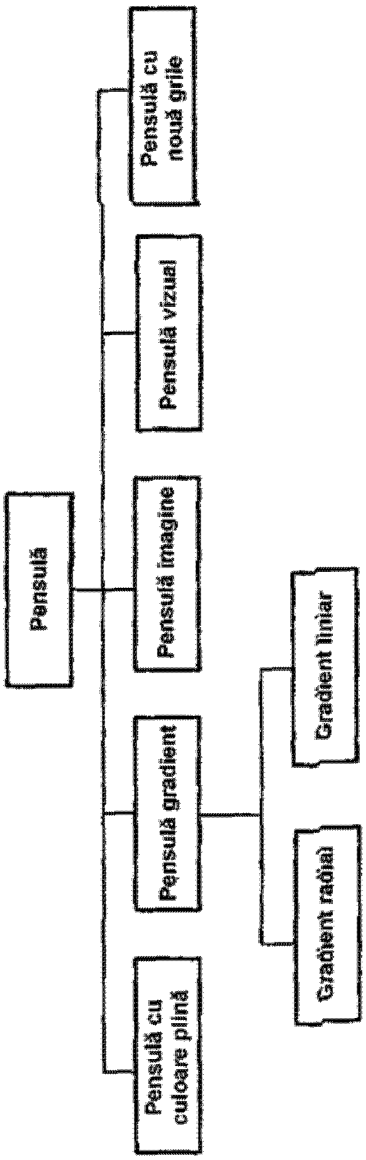

Fig. 15

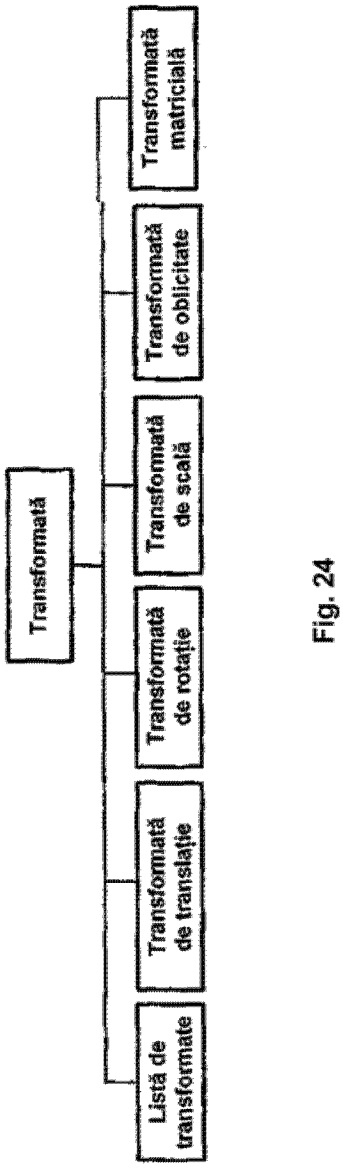

Fig. 24

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

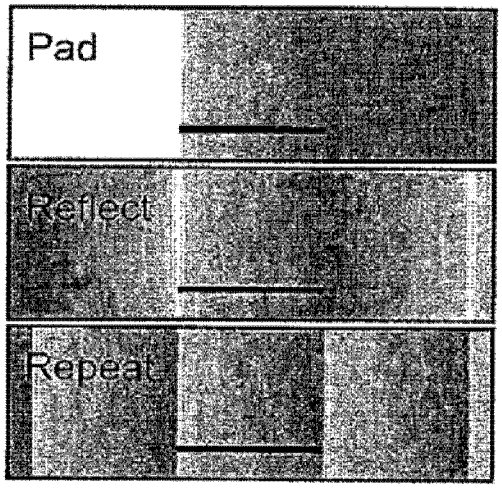

Fig. 16

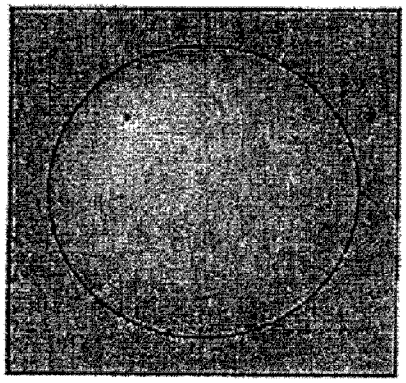

Fig. 17

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

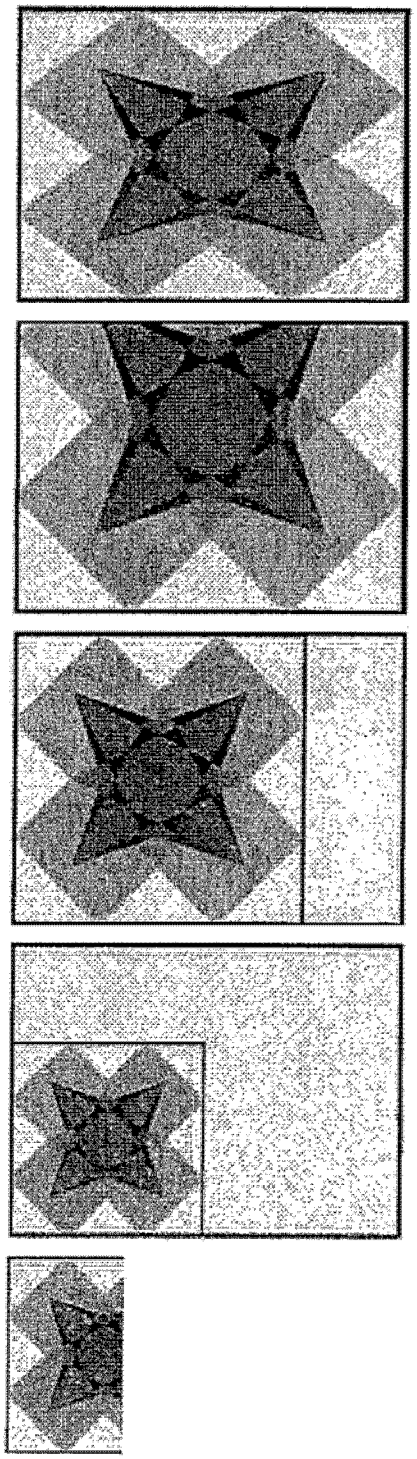

Fig. 18

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

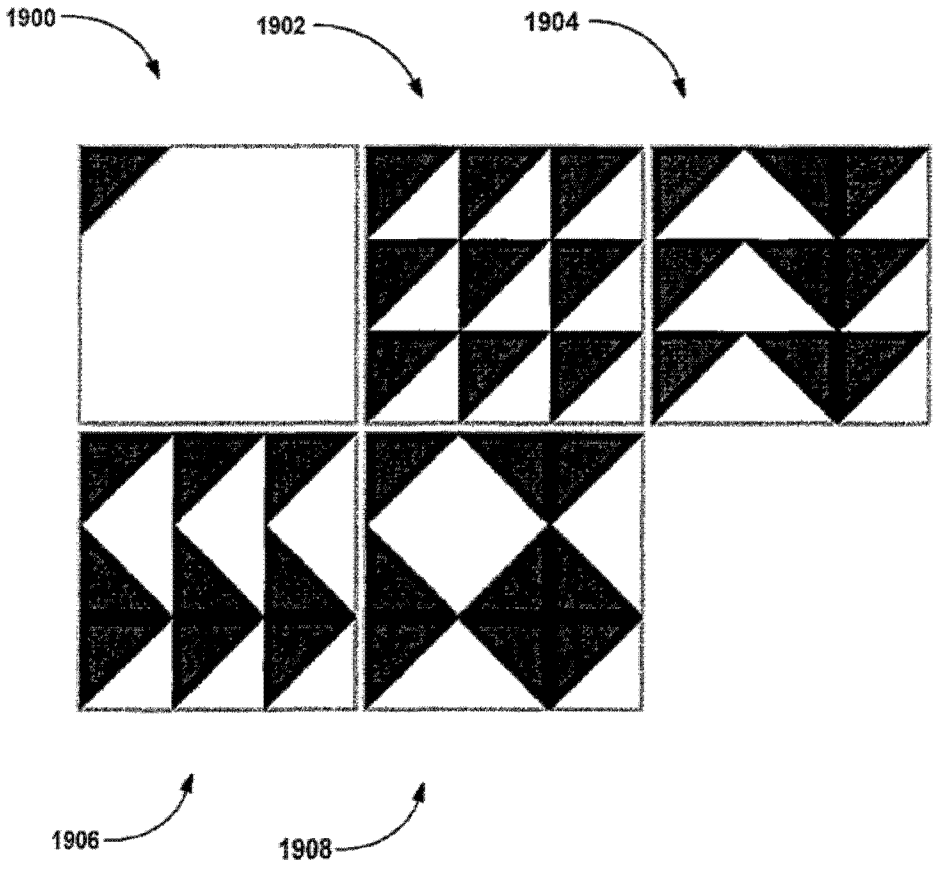

Fig. 19

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

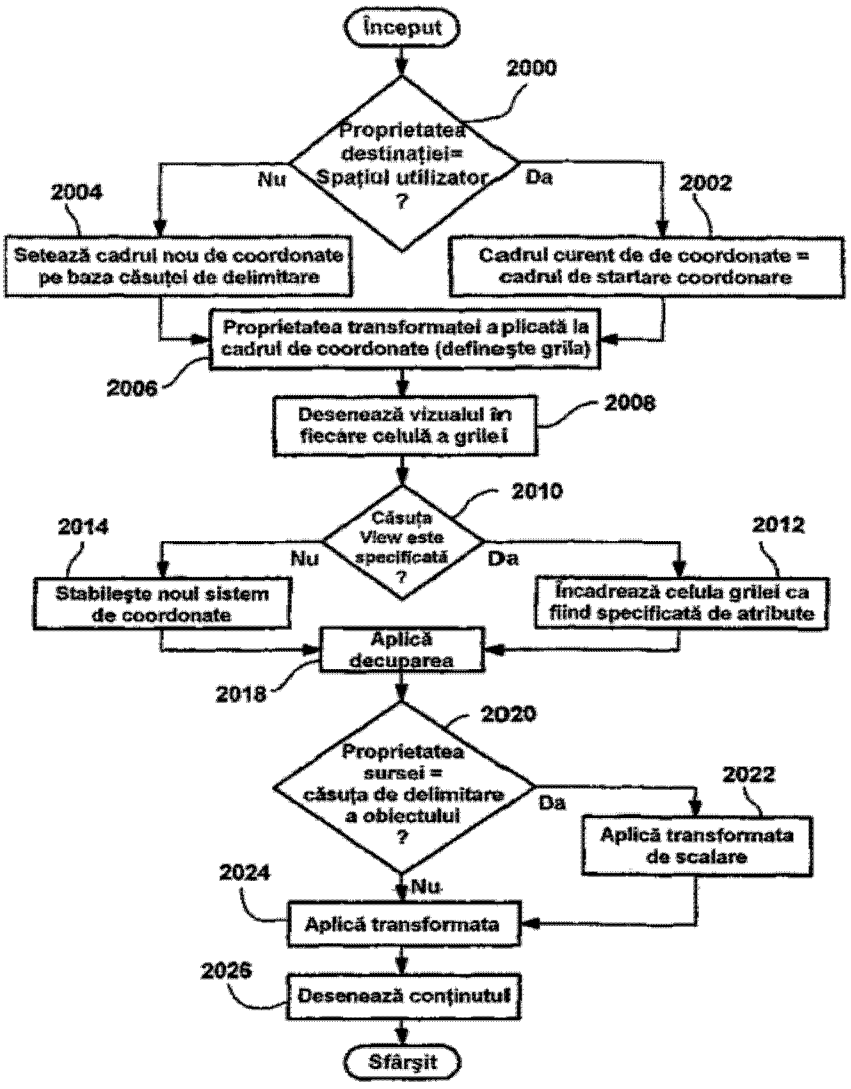

Fig. 20

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

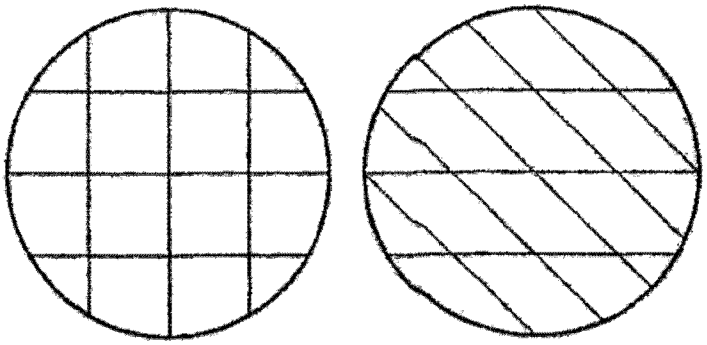

Fig. 21

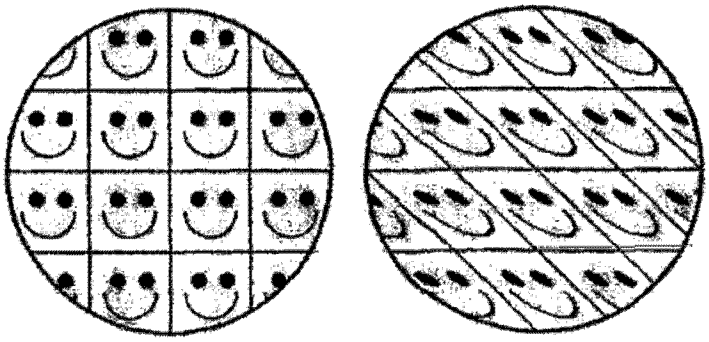

Fig. 22

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

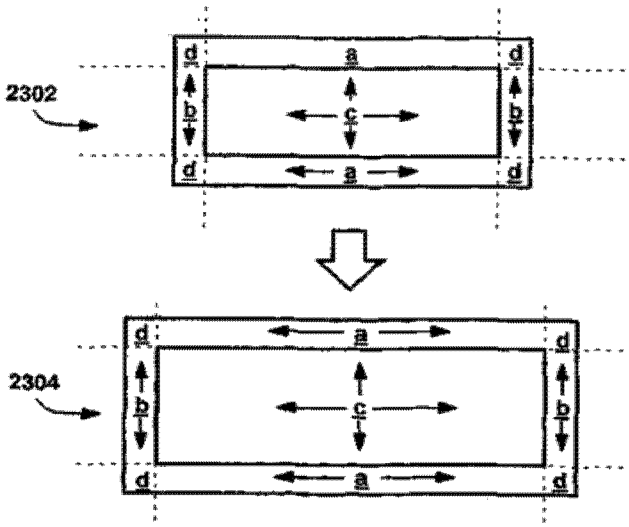

Fig. 23

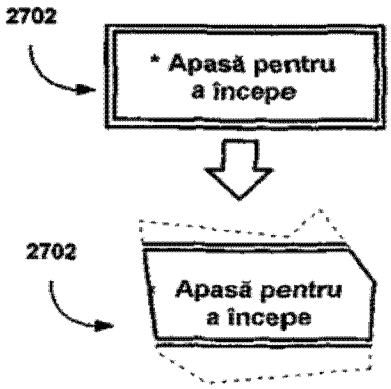

Fig. 27

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

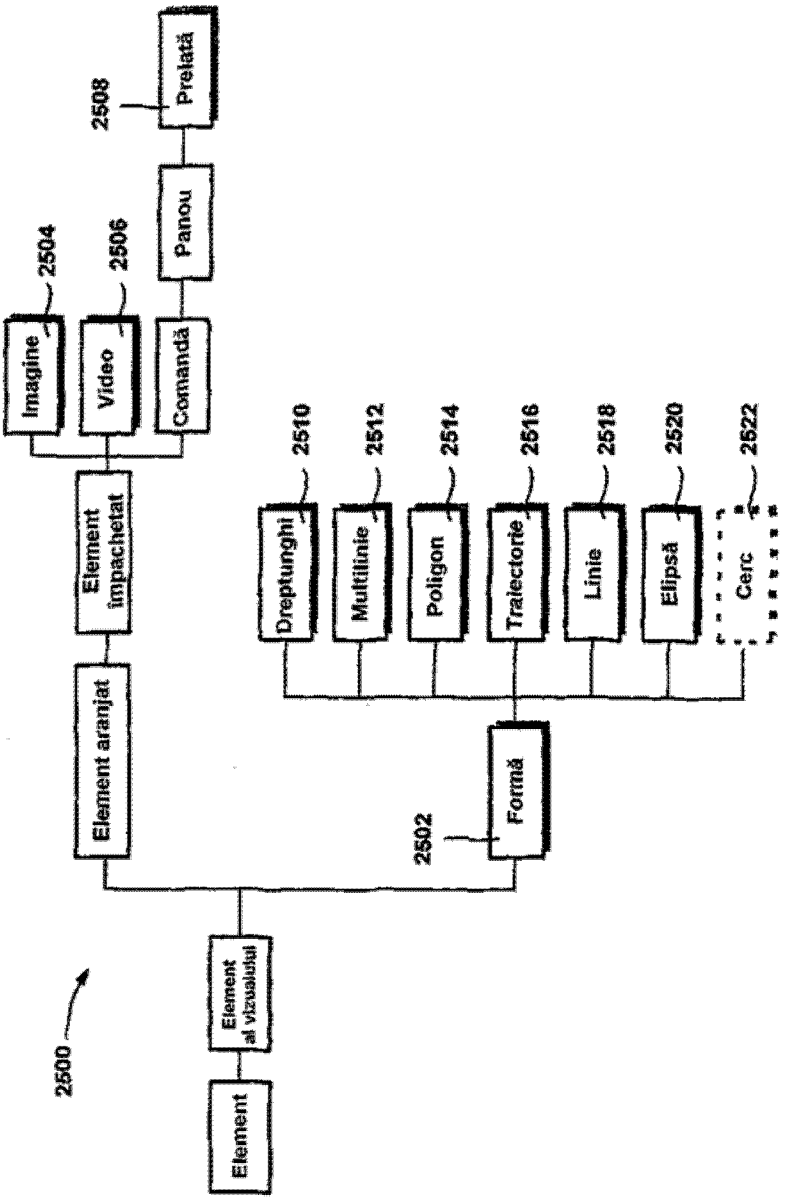

Fig. 25

(51) Int.Cl.
G06T 1/00 (2006.01);
G06T 11/00 (2006.01);
G06T 13/00 (2006.01);
G06T 15/00 (2006.01);
G06T 19/00 (2011.01);
G06F 3/048 (2006.01)

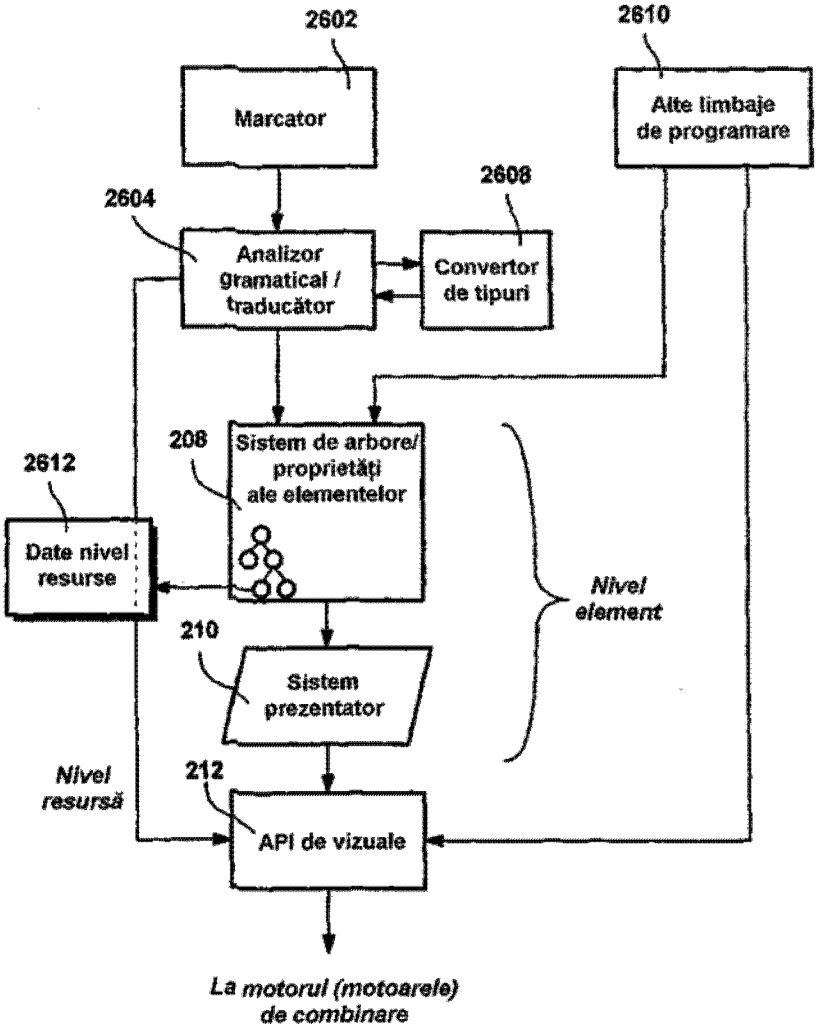

Fig. 26

Editare şi tehnoredactare computerizată - OSIM
Tipărit la: Oficiul de Stat pentru Invenţii şi Mărci
sub comanda nr. 465/2014