Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/01BE00Benka

IZAZOVI U RAZVOJU FLASH MULTIPLATFORMSKIH APLIKACIJA

CHALLENGES IN THE DEVELOPMENT OF FLASH MULTIPLATFORM APPLICATIONS

Vladislav Benka, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Ovaj rad opisuje načine renderovanja Flash aplikacija korišćenjem svojih metoda. Pokazuje na njihove procese, pravila i ograničenja. Pored toga, postoji i primer implementacije Stage3D API-ja kao i nekoliko reči o Starling Freamwork-a.*

Ključne reči: *Flash, GPU, Stage3D, Android, iOS*

Abstract – *This work describes ways to render Flash application by using its own methods. It points to their processes, rules and limitations. In addition, there is also an example of the implementation in Stage3D API, as well as a few words about Starling freamwork.*

Keywords: *Flash, GPU, Stage3D, Android, iOS*

1. UVOD

Pojavom iOS i Android mobilnih uređaja, Adobe Systems je nastavio da održava svoju poziciju u web svetu, napravivši Flash Player plugin za mobilne uređaje koji nije dobro prihvaćen od iOS Android platforme.

Zbog toga Adobe Systems proširuje svoj Adobe AIR višeplatformski sistem sa AIR SDK 2.0 bibliotekom (maja 2010.). Ovom bibliotekom AIR po prvi put ima podršku za mobilne uređaje (iOS i Android).

Pošto su u to vreme na tržištu bili mobilni uređaji sa dosta slabim CPU-om, animacije novonastalih aplikacija bile su suviše spore. Zbog toga Adobe Systems uvodi GPU mod za renderovanje ekrana sa AIR SDK 2.5 bibliotekom (oktobra 2010.). GPU mod podržan je samo za AIR aplikacije na mobilnim uređajima.

Dolaskom Adobe AIR SDK 3.0 biblioteke (oktobra 2011.), Adobe Systems uvodi novi Direct mod za renderovanje ekrana, mobilnih i desktop uređaja. Isti ovaj mod uvodi se i u Adobe Flash Player 11.

Od tada postoje tri moda za renderovanje Flash aplikacija:

- CPU mod - podržan je za sve AIR platforme, kao i u Flash Player plugin-u. U ovom modu renderovanje ekrana vrši se preko CPU-a.
- GPU mod - podržan je samo za AIR mobilne platforme (iOS i Android). Kompozicija se vrši uz pomoć GPU-a, a rasterizovanje se vrši preko CPU-a.
- Direct mod - podržan je za sve AIR platforme, kao i u Flash Player plugin-u. U ovom modu renderovanje je u potpunosti direktno preko GPU-a.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dragan Ivetić, red. prof.

2. CPU MOD ZA RENDEROVANJE EKRANA

U CPU modu renderovanje ekrana se vrši u potpunosti preko CPU-a, metodom DisplayList. Ova metoda pokazala je odlične performanse od samog početka i postala konkurentna u web svetu. Ona se zapravo svodi na iscrtavanje samo dela ekrana, dela koji se menja. Proces renderovanja sadrži tri osnovna koraka, a to su:

- Calculating dirty regions. Renderovanje započinje skeniranjem DisplayList-a, u potrazi za objektima markiranih kao Dirty (prljavi). To su zapravo objekti kojima se izmenila neka od vizuelnih osobina kao što su pozicija (x,y), nagib (rotation), veličina (scale) ili neki Blending filter. Posle skeniranja, kreira se najviše do tri pravougaona kontejnera koji sadrže Dirty objekte. Ove kontejnere nazivamo još Dirty Region i to su zapravo delovi ekrana koje je potrebno ponovo iscrtati.
- Rendering Dirty Region. Za svaki Dirty Region, Flash Player treba da utvrdi šta treba uraditi sa novim pikselima unutar kontejnera. Za ovaj postupak, kreira se scena u internom baferu zvanom Display Buffer i tu postoje sledeće faze:
  - Building edges from DisplayObjects. Flash Player ponovo skenira DisplayList, tražeći objekte koji se preklapaju sa Dirty Region-om. Rastavlja se svaki nađen objekat u set ivica i ispuna (ovo mogu biti čiste boje, gradijentne boje ili slike). Ako objekat u sebi sadrži druge objekte, ovaj proces se izvršava ponovo unutar child objekta i tako rekurzivno.
  - Rasterizing Edges. Flash Player konvertuje ivice i ispune u piksele. Dirty Region se deli na skup horizontalnih linija zvanih Scan Lines, za svaki vertikalni piksel. Sada Flash Player prolazi ovim linijama, piksel po piksel i određuje njihovu boju zasnovanu na ispunu tog područja između ivica koje ih vežu. Vreme rasterizovanja zavisi od veličine Dirty Region-a (broja piksela) i kompleksnosti objekta (objekat u objektu).
  - Applying Filters. Na kraju ovog koraka dodaju se filteri kao što su senka, sjaj, itd. Oni se dodaju na rasterizovanu sliku, kreirajući novu sliku. Postoji i mogućnost da objekat u sebi ima child objekat na kome je primenjen filter. U tom slučaju child objekat se prvo mora rasterizovati, pa primeniti filter i kao rezultat dobijamo sliku child objekta koji se koristi za određivanje ivica i ispuna parent objekta (Building edges from DisplayObjects).
- Copying to Scene - na kraju kada renderovanje prođe, Flash Player kopira piksele sa Display Buffer-a na ekran. Vreme izvršavanja ovog koraka zavisi od broja

piksela, kao i od neželjenog zaključavanja ekrana kojim upravlja operativni sistem uređaja.

Najbolja praksa kod stvaranja aplikacije korišćenjem CPU moda jeste pridržavanje određenih pravila, a to su:

- Korišćenje manjeg broja objekata koji se animiraju.
- Smanjivanje dela ekrana (dirty regions) gde se animacija izvršava, a koju CPU treba da iscrta.
- Smanjiti kompleksnost objekata, tj. da objekti nisu sastavljeni od objekata koji su sastavljeni od objekata i tako dalje.
- Smanjiti korišćenje ili potpuno ne koristiti alpha (transparentnost) svojstva objekta.

3. GPU MOD ZA RENDEROVANJE EKRANA

GPU mod je pre svega optimizovan za brzo pomeranje bitmap-a, pa stoga radi maksimalnog ubrzanja i poboljšanja performansa aplikacije, treba shvatiti kako konvertovati, do tad teške, vektorske objekte u brze rasterizovane bitmap-e.

U Flash-u postoje dve metode za konvertovanje postojeće vektorske grafike u bitmap-e, a to su:

- Caching metoda - keširanjem vektora kao bitmap-a,
- Draw metoda - crtanjem vektora kao bitmap-a.

Uključivanjem MovieClip-ovog svojstva cacheAsBitmap dobijamo da se MovieClip posmatra kao bitmap, bez obzira šta se unutar njega nalazi. Sada više nije potrebno ponovno rasterizovanje MoviClip-ovog sadržaja za vreme njegovog pomeranja. Kod za keširanje MovieClip-a je vrlo jednostavan i prikazan na Listingu 1.

```
vektorMC.cacheAsBitmap = true;
```

Listing 1. *Uključivanje svojstva cacheAsBitmap*

Keširanje u velikoj meri poboljšava performanse, ali uz sledeća upozorenja. Ako se unutar MoviClip-a koji keširamo vrši neka animacija, Flash Player će morati ponovo da rasterizuje i kešira MovieClip. Pošto je proces keširanja sistemski intenzivan, to će stvoriti negativan efekat u performansama aplikacije. Ovako keširan MovieClip je spreman samo za pomeranje po x i y osi.

Kod skaliranja i rotiranja MovieClip-a dolazi do ponovnog zahteva za rasterizovanjem i keširanjem. Zato postoji još jedno svojstvo MovieClip-a koje treba uključiti radi dobijanja performansi kod transformisanja keširanog MovieClip-a.

```
vektorMC.cacheAsBitmap = true;
vektorMC.cacheAsBitmapMatrix = new
Matrix();
```

Listing 2. *Uključivanje svojstva cacheAsBimapMatrix*

Sa ovim uključenim svojstvom, keširan MovieClip može da se skalira, rotira i menja svoju transparentnost, bez zahteva za ponovnim keširanjem. CacheAsBitmapMatrix svojstvo postoji samo u GPU render modu.

Crtanje vektora kao bitmap-a (Draw metoda) je proces u kome se uz pomoć koda vrši preslikavanje podataka piksela vektorse grafike u BitmapData objekat i kreiranje bitmap-a na kraju.

```
import flash.display.BitmapData;
import flash.display.Bitmap;

var bmd:BitmapData = new BitmapData(vektorM
C.width, vektorMC.height, true, 0x00000000)
bmd.draw(vektorMC);
var bmp:Bitmap = new Bitmap(bmd);
addChild(bm);
```

Listing 3. *Crtanje vektora kao bitmap-a*

Prvo se kreira BitmapData objekat sa dimenzijom koja obuhvata vektorski MovieClip, sa transparentnim svojstvom i transparentnom pozadinskom bojom. Posle se poziva metoda draw koja kreira podatke za bitmap iz vektorskog MovieClip-a. Na kraju se kreira bitmap objekat uz pomoć novonastalog objekta BitmapData i dodajemo ga na DisplayList-u radi prikazivanja.

Draw metoda počinje sa koordinatama (0,0) vektorskog MovieClip-a, a skaliranje i rotiranje vektorskog MovieClipa ne utiče na draw metod, što znači da se uvek dobija bitmap vektorskog MovieClipa kakav on zapravo jeste.

BitmapData objekat koristi jedinstvenu referencu, što znači da je moguće imati jednu kopiju BitmapData u memoriji i poslati te podatke na bezbroj Bitmap objekata koje se nalaze na ekranu, bez da se za njih zauzima dodatna memorija.

Za Caching metod, Flash Player automatski oslobađa memoriju keširanog vektorskog objekta kada se on više ne nalazi na ekranu, dok se kod Draw metode, memorija BitmapData objekata mora osloboditi pomoću koda. Koristi se metoda dispose(), a preporučuje se i nuliranje reference objekta.

```
bmd.dispose();
bmd = null;
```

Listing 4. *Oslobađanje memorije BitmapData-e*

Aplikacije koje koriste GPU mod za renderovanje koriste više RAM memorije od aplikacija koje rade u CPU modu, stoga se mora jako voditi računa o kreiranju bitmap-a i njegovog oslobađanja iz memorije.

Kod GPU moda za renderovanje postoje nekoliko ograničenja, a to su:

- Ako GPU ne može da renderuje objekat, on se ne prikazuje, a takođe ne postoji mogućnost prebacivanja u CPU mod radi njegovog renderovanja.
- Blend modovi, kao što su: layer, alpha, erase, overlay, hardlight, lighten i darken, nisu podržani.
- Filteri na objektima nisu podržani.
- PixelBlender nije podržan.
- GPU renderovanje je nedostupno za stare android uređaje koji imaju slabu GPU jedinicu.

GPU mod je efikasan samo za bitmap-e, jednostavne vektore i display objekte koji imaju uključena svojstva cacheAsBitmap i cacheAsBitmapMatrix. Najbolja praksa kod stvaranja aplikacije u GPU modu jeste pridržavanje sledećih pravila:

- Ograničenje broja vidljivih objekata na stage-u.
- Ponovno korišćenje objekata.
- Korišćenje bitmap-a rezolucije od 2^n x 2^m.
- Uključivanje svojstva cache za vektorske objekte koji se ne menjaju često.
- Skaliranje bitmap-a na željenu dimenziju pre uvoženja u samu aplikaciju.
- Spuštanje parametra za stage kvalitet (LOW).

GPU mod je alternativa za renderovanje aplikacija na mobilnim uređajima uz potencijal pružanja vrhunske performanse, ali u isto vreme i teži način za kodiranje.

4. DIRECT MOD ZA RENDEROVANJE EKRANA

Pre pojave Direct moda za renderovanje, u Flesh-u su se razvijale 3D aplikacije pomoću softverskih 3D engine-a, gde se renderovanje vršilo preko CPU-a, i kao rezultat toga, aplikacije su bile jako spore.

Pojavom Direct moda za renderovanje i Stage3D API-a (kodno ime Molehill), 3D sadržaj se renderuje u realnom vremenu, direktno uz pomoć GPU-a.

Stage3D je API sa niskim nivoom pristupa GPU API-ju, posebno dizajniran da iskoristi maksimalnu snagu hardverskih 3D grafičkih jedinica uređaja. Neverovatno je brz, jednostavan i služi isključivo za 3D renderovanje.

Prebacivanjem procesa renderovanja na GPU jedinicu, oslobađa se CPU jedinica koja sad može da obavlja druge proračune. Ovo otvara mogućnost za kompleksnije aplikacije pošto su sada CPU i GPU 100% u upotrebi.

Kako 3D grafika funkcioniše, objašnjeno je u sledećem delu. Generalno, 3D scena definisana je grupom 3D geometrijskih objekata (mesh). 3D geometrijski objekat predstavljen je kao skup poligona (polies). Jedan poligon može biti sastavljen od jednog ili više trouglova (triangles), a svaki trougao sačinjen je uz pomoć tri tačaka (vertices). Dakle, 3D scena je skup definisanih tačaka i eventualno dodatnih informacija za renderovanje, kao što su boja ili tekstura.

Kod Stage3D API-a, podaci 3D scene se definišu i prosleđuju u GPU. Oni se smeštaju u GPU memoriju gde se obrađuju, trougao po trougao. Kao rezultat dobija se finalna renderovana slika, spremna za prikaz na ekranu.

3D renderovanje vrši se u GPU jedinici uz pomoć 3D programibilnog renderskog pipeline-a. 3D renderski pipeline je proces renderovanja, podeljen u više elementarnih operacija. Sastoji se od niza blokova, gde svaki blok izvršava jednu elementarnu operaciju, a postavljeni su kaskadno, tako da je izlaz jednog bloka ulaz sledećeg bloka.

Programibilni pipeline znači da podržava Shaders programe. To su mali programi koji se pokreću u GPU-u, pomoću kojih je moguće uticati na tok renderovanja 3D scene. Ovi Shader programi se, takođe, prosleđuju na GPU. Postoje dva tipa Shader programa, a to su:

- Vertex Sahader - programi koji utiču na transformaciju položaja tačaka.
- Fragment Shader - programi koji utiču na transformaciju boja trouglova.

Adobe Systems kreirao je dva Shading jezika za kreiranje programa:

- AGAL (Adobe Graphics Assembly Language) jezik - asemblerski jezik niskog nivoa.
- Pixel Blender 3D - jezik višeg nivoa, lakši za razumevanje, ali teži za potpunu kontrolu.

5. PRIMER STAGE3D API-JA

U sledećem delu je primer za prikaz obojenog kvadrata, korišćenjem Stage3D API-ja. Primarna klasa Stage3D API-ja jeste klasa Context3D, koja služi kao površina za 3D renderovanje. Zato je, na početku aplikacije, potrebno definisati event listener za CONTEXT3D_CREATE i ujedno pozvati zahtev za njeno kreiranje.

```
stage.stage3Ds[0].addEventListener(Event.CO
NTEXT3D_CREATE, configContext3D);
stage.stage3Ds[0].requestContext3D();
```

Listing 5. *Kreiranje Context3D-a*

Čim se Context3D kreira, vrši se njegova konfiguracija pozivom metode configureBackBuffer uz parametre:

- širine i visine površine za renderovanje,
- minimalnim nivoom anti-aliasing-a i sa
- parametrom true, za kreiranjem depth i stencil bafera.

```
var context:Context3D;
context=stage.stage3Ds[0].context3D;
context.configureBackBuffer(stage.StageWidt
h, stage.StageHeight, 0, true);
```

Listing 6. *Konfiguracija Context3D-a*

Posle konfiguracije Context3D-a, definišemo 3D geometrijski objekat, u ovom primeru obojeni kvadrat, koji sadrži četiri tačke i svaka od njih ima svoju boju. To činimo definisanjem Vertex Buffer-a, strukture koja sadrži sve informacije o tačkama geometrijskog objekta. Ovaj Vertex Buffer sadržaće 6 Vertex atributa. Tri atributa za položaj (x,y,z) i tri atributa za boju (r,g,b). Definisan Vertex Buffer držimo u vektor nizu. Zatim kreiramo VertexBuffer3D instancu (za 4 tačke sa po 6 atributa) i preko nje otpremamo Vertex Buffer na GPU.

```
var vertices:Vector.<Number>=
Vector.<Number>([
-0.3,-0.3,  0,  1,  0,  0, // x,y,z,r,g,b
-0.3, 0.3,  0,  0,  1,  0,
 0.3, 0.3,  0,  0,  0,  1,
 0.3,-0.3,  0,  1,  0,  1
]);
var vBuff:VertexBuffer3D;
vBuff=context3D.createVertexBuffer(4, 6);
vBuff.uploadFromVector(vertices, 0, 4);
```

Listing 7. *Definisanje VertexBuffer-a Context3D-a*

Pošto smo definisali Vertex Buffer, sledeći korak je definisanje Index Buffera. Geometrijski objekat se rastavlja na skup trouglova. Ovde se kvadrat rastavlja na dva trougla koja se definišu pomoću Index Buffer-a. Redosled unosa tačaka trougla mora biti u smeru kazaljke na satu, da bi renderovao lice slike.

Kao kod Vertex Buffer-a i ovde se Index Buffer čuva u vektor nizu. Kreiramo IndexBuffer3D instancu (2 trougla, 6 tačaka) i preko nje otpremamo Index Buffer na GPU.

```
var indices:Vector.<uint>=
Vector.<uint>([0, 1, 2, 2, 3, 0]);
var iBuffer:IndexBuffer3D;
iBuffer=context3D.createIndexBuffer(6);
iBuffer.uploadFromVector(indices, 0, 6);
```

Listing 8. *Definisanje IndexBuffer-a Context3D-a*

Sada su nam potrebni još Vertex i Fragment Shader-i. Vertex Shader jednostavno transformiše položaje tačaka prema transformacionoj matrici koju definišemo ActionScript-om, dok Fragment Shader transformiše boju. Korišćenjem AGAL-a kreiramo kod za Shader-e i pomoću Program3D klase otpremamo Shader-e na GPU.

```
var shaderP:Program3D;
var vShader:AGALMiniAssembler=new
AGALMiniAssembler();
vShader.assemble(
    Context3DProgramType.VERTEX,
    "m44 op, va0, vc0\n" + // pos
    "mov v0, va1" // copy color
);
var fShader:AGALMiniAssembler = new
AGALMiniAssembler();
fShader.assemble(
```

```
    Context3DProgramType.FRAGMENT,
    "mov oc, v0"
);
shaderP=context.createProgram();
shaderP.upload(vertexShader.agalcode,
fragmentShader.agalcode);
```

Listing 9. *VertexShaderar i FragmetShader*

Scena je sada spremna za rederovanje i preostala je implementacija funkcije za renderovanje. Ona se izvršava u svakom frejmu na sledeći način:

- Prvo se poziva metoda **Context3D::clear** koja setuje sve piksele (Color Buffer) na boju koju prosleđujemo. U primeru (1,1,1,1), to je bela boja bez transparencije.
- Setuju se Vertex Buffer-i, tj. tačke i boje. Tačke se setuju na vertex atributsku lokaciju 0 (va0), a boje na vertex atributsku lokaciju 1 (va1).
- Prosleđuje se Shader Program.
- Prosleđuje se transformaciona matrica radi menjanja položaja našeg geometrijskog objekta, pomoću metode **Context3D::setProgramConstantsFromMatrix**.
- Poziva se metoda **Context3D::drawTriangles** koja renderuje trouglove na površini za renderovanje.
- Na kraju kada smo završili sa renderovanjem našeg geometrijskog objekta poziva se metoda **Context3D::present,** koja govori Stage3D-u da je aplikacija završila sa renderovanjem i da je spremna da se prikaže na ekranu (Color Buffer).

```
context.clear(1, 1, 1, 1);

context.setVertexBufferAt(0,vBuff,0,Context
3DvertexBufferFormat.FLOAT_3);

context.setVertexBufferAt(1,vBuff,3,Context
3DVertexBufferFormat.FLOAT_3);

context.setProgram(sProgram);

var mat:Matrix3D = new Matrix3D();
mat.appendRotation(getTimer()/50,
Vector3D.Z_AXIS);

context.setProgramConstantsFromMatrix(Conte
xt3DProgramType.VERTEX, 0, mat, true);
context.drawTriangles(iBuff);

context.present();
```

Listing 10. *Funkcije za renderovanje*

6. STARLING FREAMWORK

Starling je ActionScript 3.0, 2D Framework, razvijen na osnovu Stage3D API-ja. Omogućava brz razvoj GPU akcelerovanih AIR aplikacija, bez direktnog korišćenja Stage3D API-ja. Starling koristi Stage3D API, koji je zapravo GPU API niskog nivoa, pokrenut preko OpenGL-a i DirectX-a na desktop platformi ili preko OpenGL-a ES2 na mobilnoj platformi.

Starling pojednostavljuje složenost niskog nivoa Stage3D API-ja i omogućava lako i intuitivno programiranje za sve. Jednostavan je i lak za učenje, pogotovo za Flash programere koji su već upoznati sa ActionScript konceptom. Starling API u sebi sadrži dosta klasa sa istim imenima i namenama kao u ActionScript API-ju.

Mnogi smatraju da je Stage3D API striktno namenjen za 3D sadržaje, međutim, u stvarnosti je moguće kreirati i 2D sadržaj. GPU je veoma efikasan kada se radi o iscrtavanju trouglova (triangles), tako da pomoću drawTriangles API-ja iscrtavamo dva trougla (kvadrat) na koje prilepimo teksturu, korišćenjem UV mapiranja. Takav objekat predstavlja se kao Starling Sprite, koji ima većinu osobina kao ActionScript Sprite. Napomena jeste da ako ovom Sprite-u menjamo teksturu u svakom frejmu, dobijamo animirani Starling-MovieClip. Ova tehnika menjanja teksture radi dobijanja animacije naziva se Sprite Sheets tehnika.

Da bi se predstavilo kako Starling smanjuje složenost Stage3D API-ja, moguće je uporediti gorenavedeni kod za prikazivanje kvadrata pomoću Stage3D API-ja i sledeći kod pisan pomoću Starling-a.

```
var tex:Texture=Texture.fromBitmap(new
embeddedBitmap());

var img:Image = new Image(texture);
img.x = 300;
img.y = 150;
addChild(img);
```

Listing 11. *Starling kod za iscrtavanje slike*

7. ZAKLJUČAK

U radu su predstavljeni 3 moda za renderovanja Flash aplikacija kao i Stage3D API, i Starling Freamwork. Tokom izrade ovog rada, vršilo se testiranje performansi ovih modova. Posmatrao se maksimalan broj grafičkih objekata koji se pomeraju na ekranu mobilnih uređaja. Ponašanje aplikacije pratio se pomoću Adobe Scout CC-a programa. Rezultati testova za iPhone 5c su sledeći:

- CPU mod, 200 objekata, 59.8fps.
- GPU mod, 700 objekata, 59fps.
- Starling Freamwork, 1200 objekata, 59fps.
- Stage3D API, 2650 objekata, 59.8fps.

Rezultati za Samsung S4 su:

- CPU mod, 20 objekata, 59.8fps.
- GPU mod, 160 objekata, 60fps.
- Starling Freamwork, 500 objekata, 58.2fps.
- Stage3D API, 2700 objekata, 60.1fps.

Ovim testovima moguće je potvrditi da Stage3D API ima sjajne performanse i da ga netreba izbegavati.

8. LITERATURA

[1] https://www.adobe.com/devnet/scout/articles/understanding-flashplayer-with-scout.html (pristupljeno u sept. 2018.)

[2] https://www.adobe.com/devnet/flashplayer/articles/how-stage3d-works.html. (pristupljeno u sep. 2018.)

[3] I. Thiabult, "Starling – Building GPU Accelerated Application ", O'Reilly Sebastopol, CA 95472, 2012.

Kratka biografija:

Vladislav Benka rođen 6. maja 1978. godine u Stuttgart, Nemačka. 1997. godine upisao je osnovne akademske studije na Fakultetu tehničkih nauka u Novom Sadu. Položio je sve ispite propisane planom i programom.