3. Organizacija podataka

- dobivene podatke (objekte) potrebno je pretvoriti u pogodan oblik za daljnje korištenje, npr:
  - podaci dobiveni uzorkovanjem mogu biti točke površine (surfel) a za daljnje korištenje bi bilo pogodno imati mrežu poligona (trokuta) https://potree.org/potree/examples/vr_eclepens.html
  - volumen podataka – može biti prikazivan izravno ili trebamo odrediti poligone koji čine površinu
- potrebno je povezati različite vrste podataka
  - vrhovi, normale, poligoni, strukture povezivanja površine objekta s kosturom, svojstva površine (materijali), teksture, fizikalna svojstva objekta, audio, dodatne strukture za određivanje kolizije, … = *multimodalni* podaci
- podatke s jedne strane želimo imati organizirane prilagođeno čovjeku a s druge strane organizirane prilagođeno računalu
  - efikasno izvođenje na računalu

Organizacija podataka

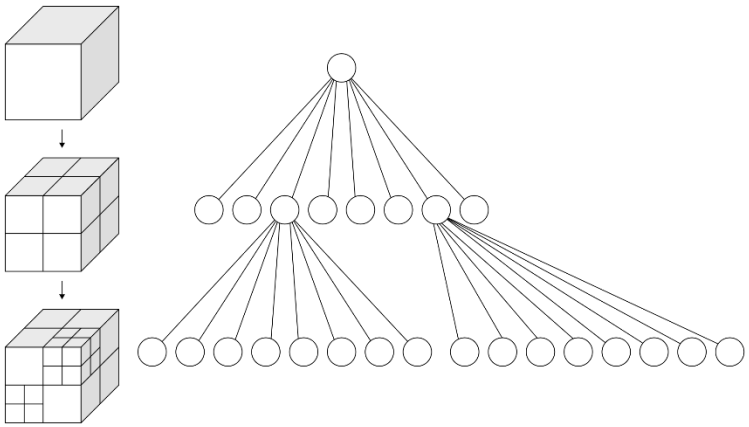
- različite organizacije podataka u cilju efikasnijeg baratanja
  - posebno važno kod aplikacija koje moraju raditi u stvarnom vremenu
  - organiziranje podataka u cjeline koje omogućuju brzo dohvaćanje podataka u različitim postupcima koji se provode
    - često je važno susjedstvo (npr. računanje normala, vidljivosti, kolizije, sjecište zrake) (https://potree.org/potree/examples/vr_eclepens.html)
  - nasljeđivanje svojstava pojedinih cjelina
    - kada definiramo svojstva materijala jedne cjeline često želimo da se ta svojstva prenesu na pod-cjeline (OpenGL- stroj stanja, DirectX OOP) zbog uštede prostora i vremena (programera i računala)
    - nasljeđivanje transformacija
  - iskorištavanje sklopovskih mogućnosti
    - npr. arhitektura protočne strukture, SIMD, MISD,
    - prilagođavanje GPU – nepoželjno je “seljenje” između CPU i GPU

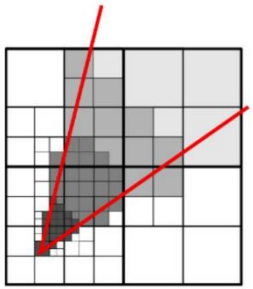
3.1. Strukture podataka

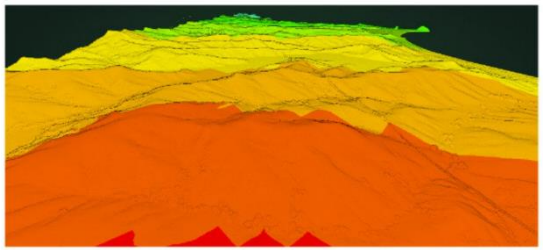
- hijerarhijska organizacija
  - strukturiranje podataka u cjeline, obično prostorno susjedne (npr. strukture stabla – BSP, octree, pokretni dijelovi objekta)
  - indeksirani pristup popisu vrhova, tekstura, normala …
  - obuhvaćanje pojedinih cjelina npr. omeđujućim volumenima, (BVH-*Bounding Volume Hierarchies*), pridjeljivanje svojstava
- iskorištavanje sklopovlja
  - nizovi istovrsnih podataka koje odjednom treba obraditi
  - http://callumprentice.github.io/apps/geometry_to_buffergeometry/index.html (Cone/Density/fps)
- često je kontradiktorno (iskorištavanje GPU i strukturiranje)
  - prilagođavanje sklopovlju (problem prenosivosti)
- **graf scene, polja podataka**

Podsjetnik struktura podataka

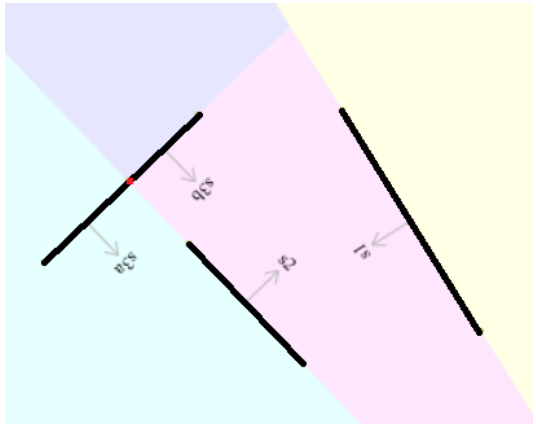
Oktalno stablo (*engl. octree*)

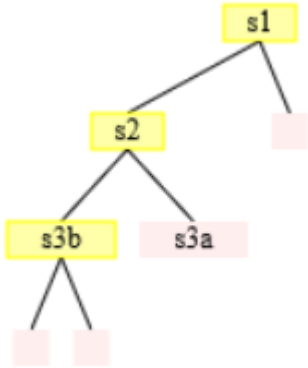




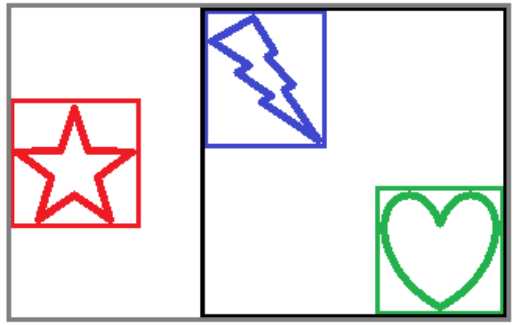


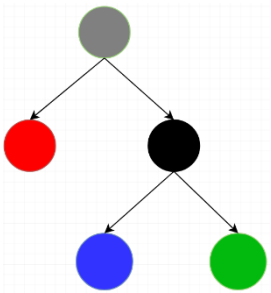
BSP stablo (*engl. binary space partition*)





BVH (*engl. Bounding volume hierarchy*)



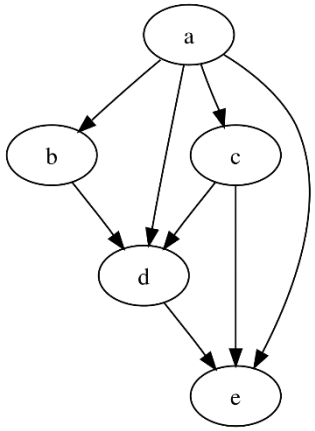


Scena i objekti

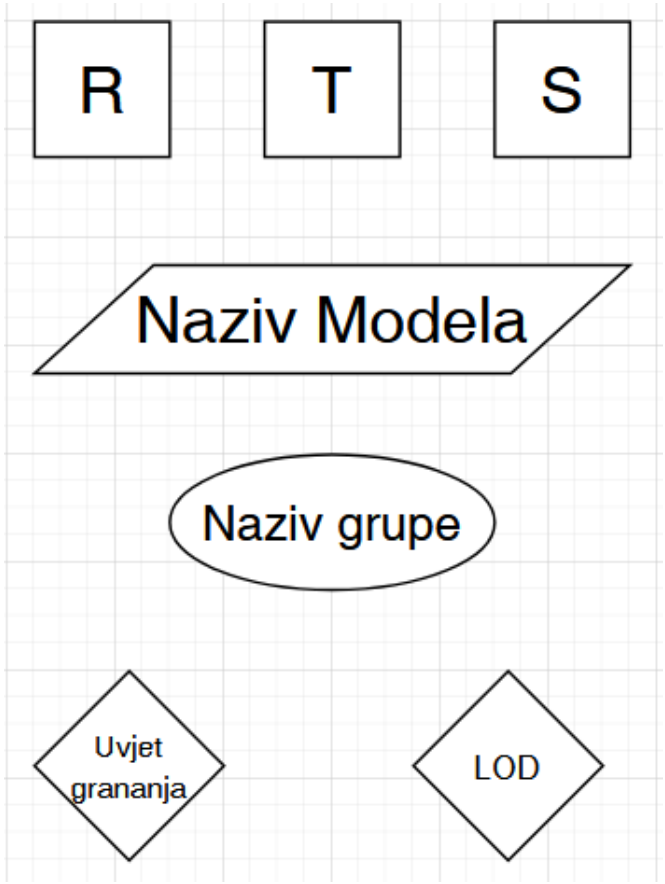
- alati za izgradnju (učitavanje) scene i pripadni zapisi datoteka
  - Maya, 3DMax, Blender3d, TerrainPro…
  - Pixar - *Universal Scene Description* https://graphics.pixar.com/usd/release/index.html
- spremnici podataka (engl. buffer object)
  - za računalo učinkovite strukture podataka
- složeni modeli
  - eksplicitno definirani
  - proceduralno definirani (fraktali, NURBS)
- graf scene
  - scena (objekti) je hijerarhijski organizirana
  - olakšano je pretraživanje i baratanje pojedinim objektima
  - detekcija kolizije i pripadna reakcija
  - promjene razine složenosti (LOD)

Graf scene (*engl. scene graph*)

- hijerarhijski način modeliranja složenih objekata, scene i organizacija redoslijeda ostvarivanja promjena i prikaza
  http://math.hws.edu/graphicsbook/source/threejs/diskworld-1.html
- acikličan usmjeren graf (nema petlji u grafu)
  - graf je struktura podataka sastavljena od čvorova i lukova
    - čvor predstavlja podatkovni element
    - luk je veza između podatkovnih elemenata
- primjeri korištenja grafa scena:
  - Unreal engine, Unity
  - OpenInventor (SGI), jMonkeyEngine, Openscenegraph, OGRE
- može se implementirati uz API niže razine
  - OpenGL, Direct3D



Grafički simboli za prikaz grafa scene



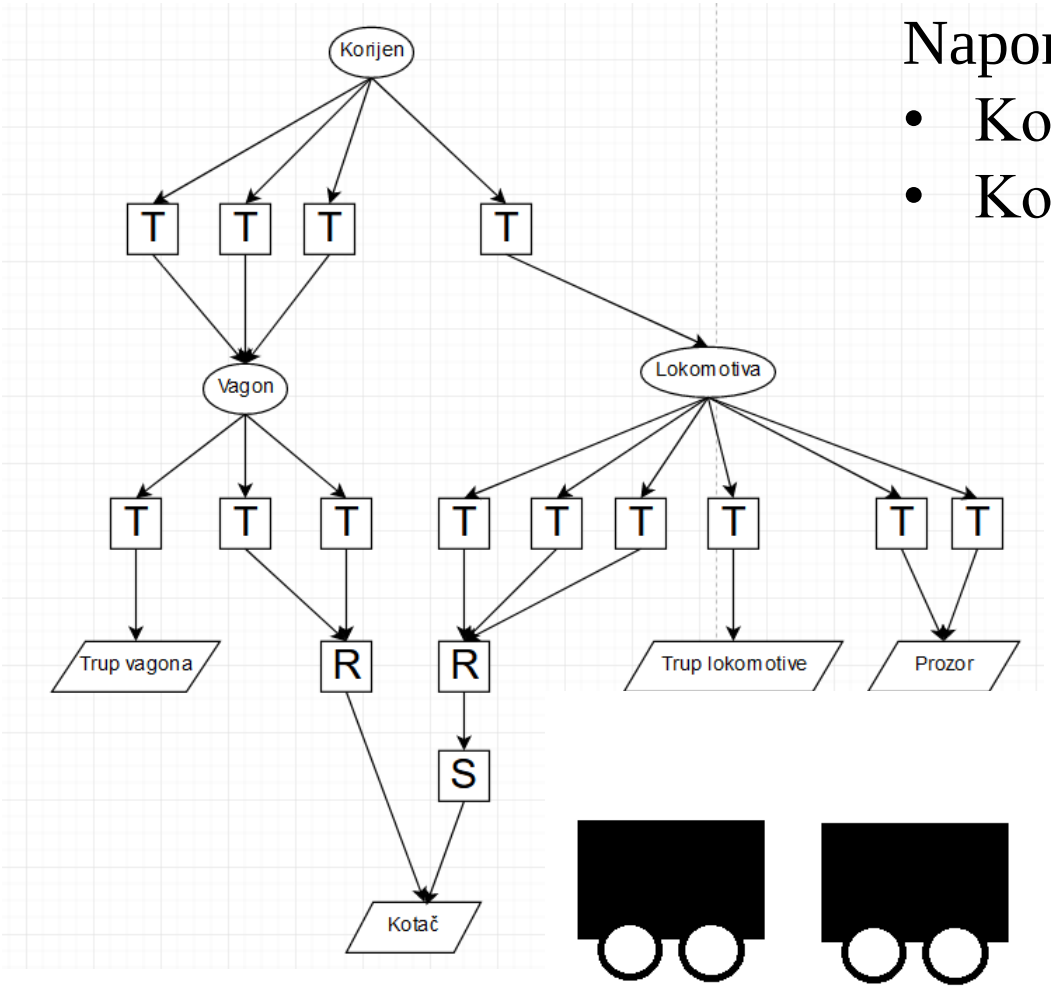
Transformacijske matrice (rotacija, translacija, skaliranje)

Objekt koji se može iscrtati

Čvor grupiranja

Čvor izbora, LOD *(level of detail)*

Primjer grafičkog prikaza scene



Napomene:

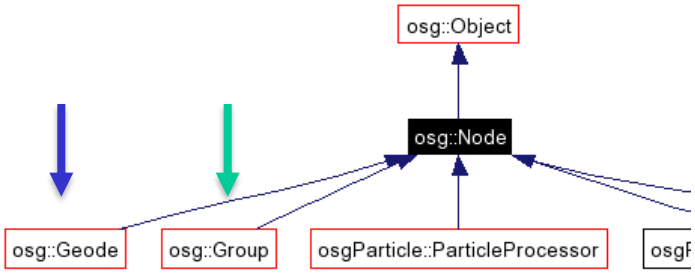
- Kotači se mogu rotirati
- Kotači na lokomotivi su veći

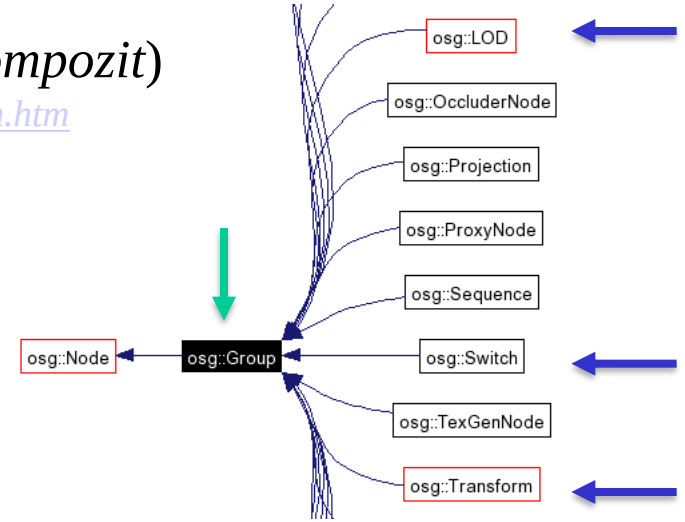
Primjer implementacije grafa scene (OpenSceneGraph)

Strelice označavaju nasljeđivanje

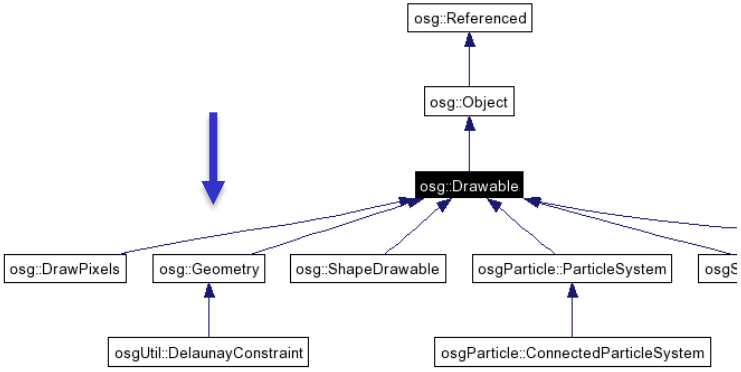
Node je čvor grafa scene (obrazac uporabe *Kompozit*)

https://www.tutorialspoint.com/design_pattern/composite_pattern.htm





Geode sadrži objekte tipa *Drawable*



Geometry sadrži podatke o mreži poligona

Transform - sadrži 4x4 matricu

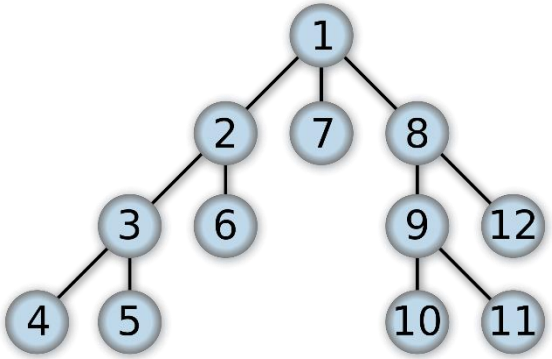
Switch - mogućnost odabira trenutno „aktivnog” djeteta

LOD - slično kao i switch.

-Automatski odabir djeteta ovisno o udaljenosti od kamere

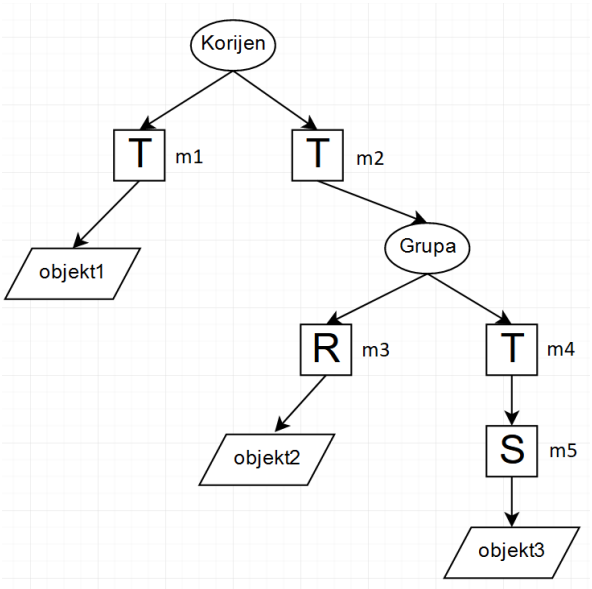
Obilazak grafa scene (OpenSceneGraph)

- Razred Viewer
  - sadrži korijen (*Node*) grafa scene
  - zadužen za obilazak grafa scene prilikom poziva metoda:
    - *update()*
      - pozivanje metoda za osvježavanje animacija i transformacija modela
      - **kompozicija transformacija**
      - određivanje vidljivih modela (*engl. Culling*)
    - *frame()*
      - pozivanje metoda za instancirano iscrtavanje prikupljenih modela u *update()* metodi
- Obilazak grafa najčešće pomoću DFS (*engl. Depth-first search*)
- Prikazivač prikazuje objekte u redoslijedu za koji utvrdi da je najučinkovitiji (višeprocesorski)



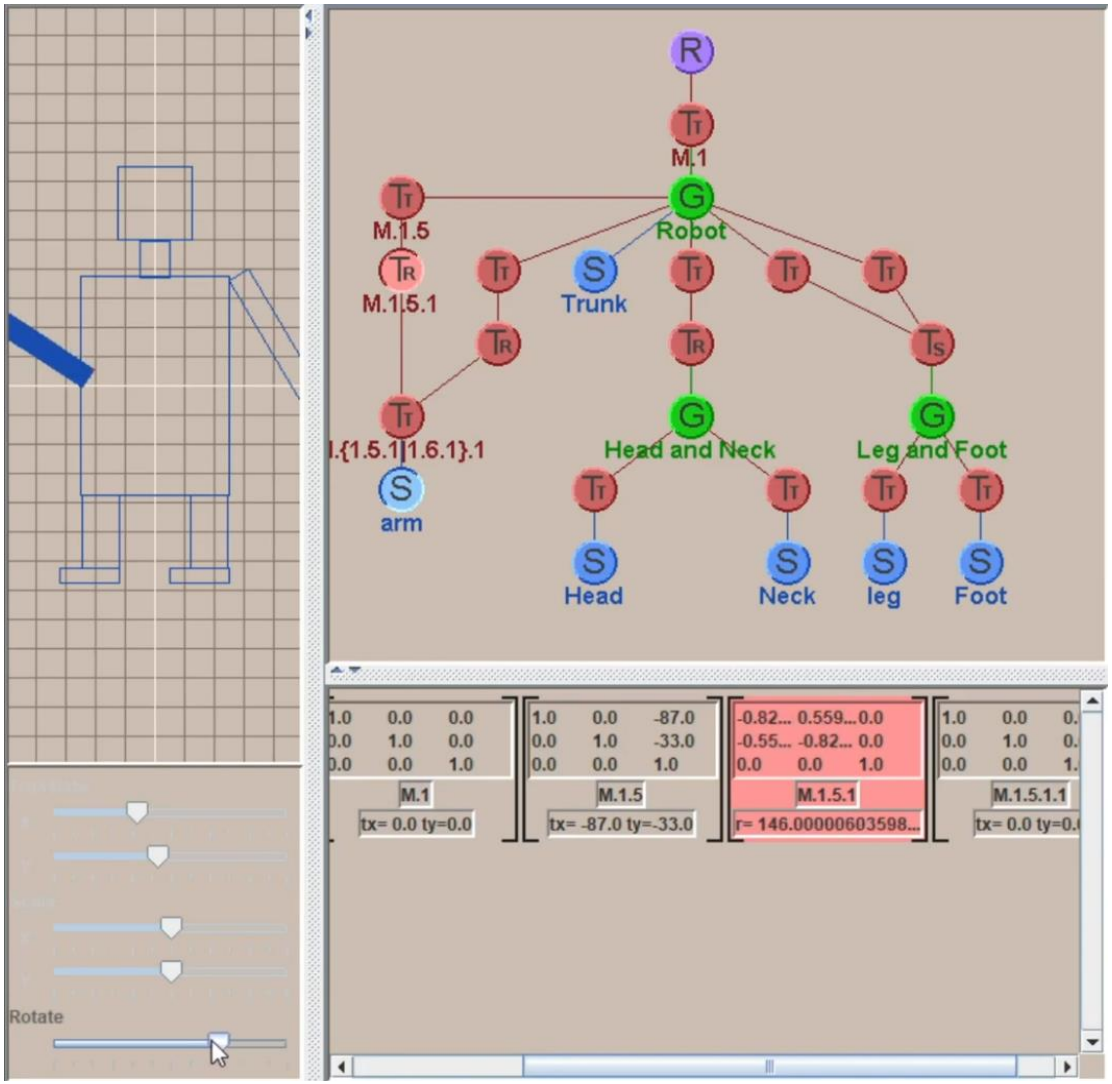
Obilazak grafa scene

- Kompozicija transformacija u grafu scene
  - transformacijski čvorovi (T, R, S) određuju transformacije koje će biti obavljene nad čvorovima u njihovom podstablu.
  - CTM (*engl. composite transformation matrix*) je ukupna transformacijska matrica koju određuju sve transformacije čvorova roditelja za pojedini čvor
  - transformacija roditelja utječe na cijelo pod-stablo
  - (moguće su različite implementacije)
  - CTM o1 – m1 v_o
  - CTM o2 – m2 m3 v_0
  - CTM o3 – m2 m4 m5 v_o



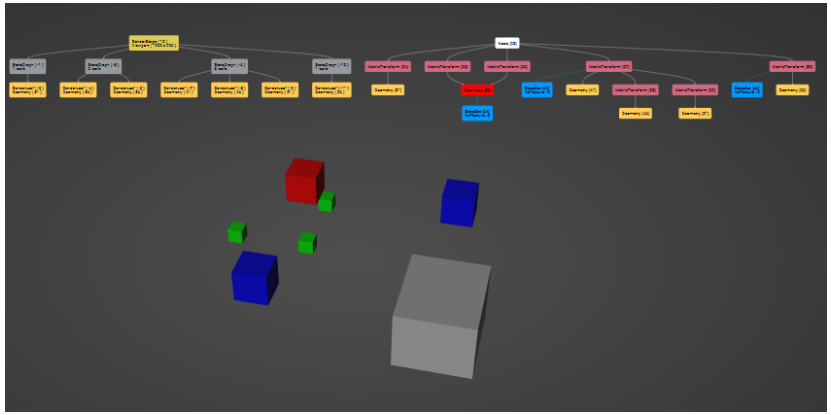
http://cs.wellesley.edu/~cs307/threejs/demos/BasicModeling/TeddyBear-composite.shtml

Obilazak grafa scene - primjer

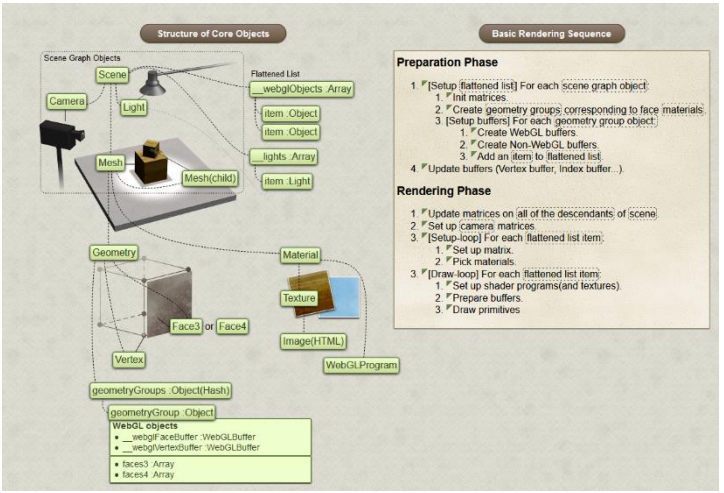


CTM S_{arm} – M.1* M.1.5*M.1.5.1 * M.1.5.1.1* v_o

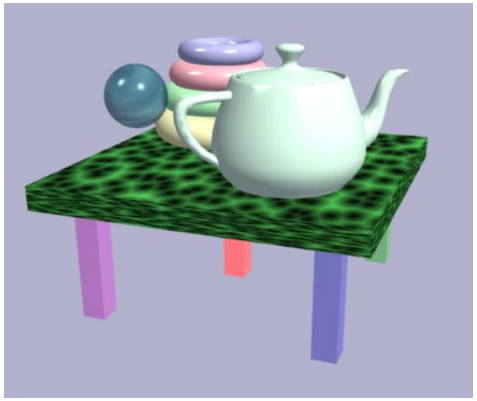
Primjeri grafa scene

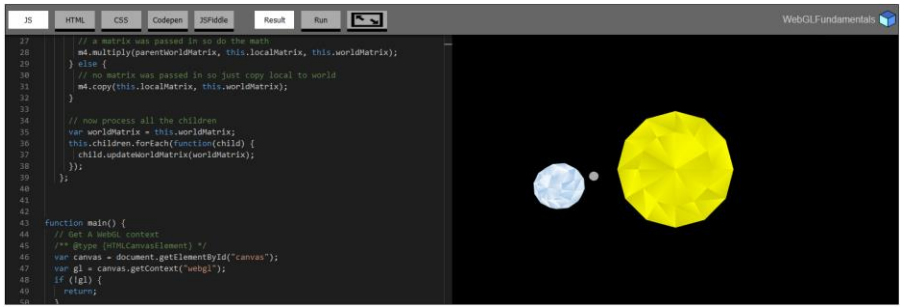


https://cedricpinson.github.io/osgjs-website/examples/scene-debug/



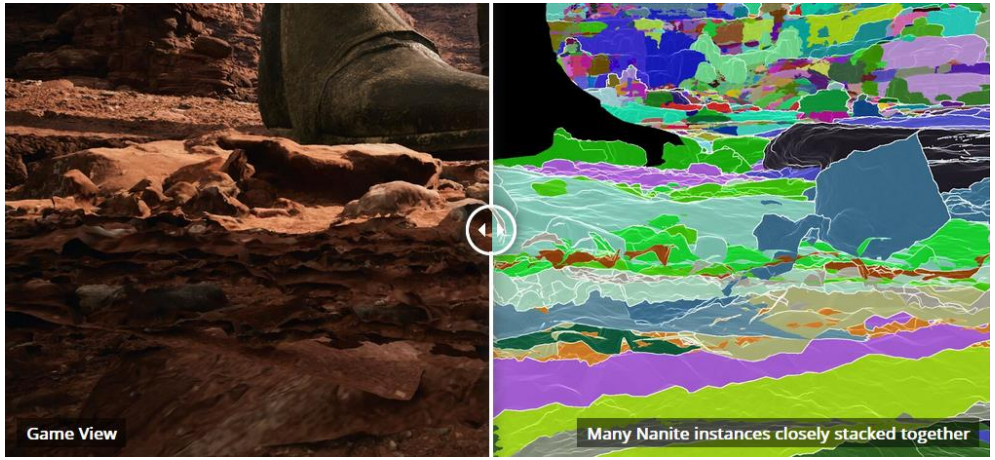
http://ushiroad.com/3j/

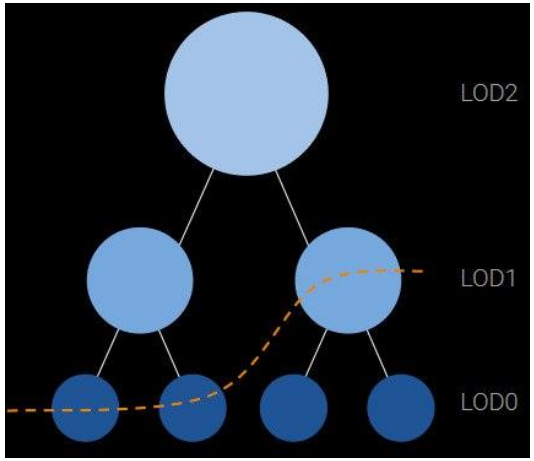




http://webglfundamentals.org/webgl/lessons/webgl-scene-graph.html

Primjeri grafa scene – UE5 Nanite

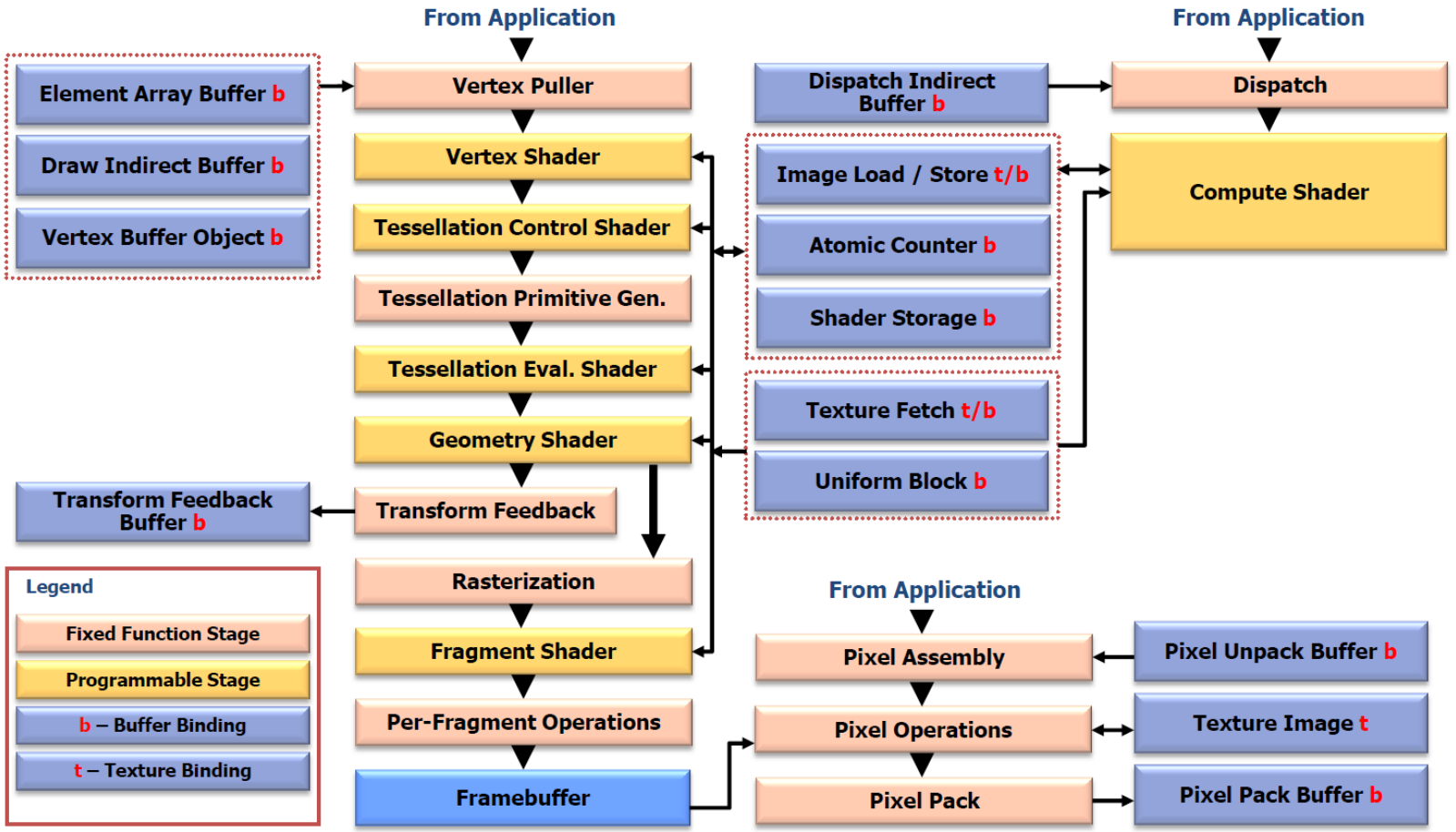




- Tijekom izgradnje scene
  - Automatska izgradnja čvorova za potrebe promjene razine složenosti (LoD)
  - Problem trganja mreže poligona prilikom korištenja čvorova s različitih razina stabla
- Tijekom prikaza scene
  - Automatski odabir čvorova za prikaz na temelju pozicije kamere
  - Broj pixela ~ broj trokuta za prikaz
- https://docs.unrealengine.com/5.0/en-US/nanite-virtualized-geometry-in-unreal-engine/

3.2. Organizacija podataka u OpenGL-u

Podsjetnik OpenGL protočnog sustava



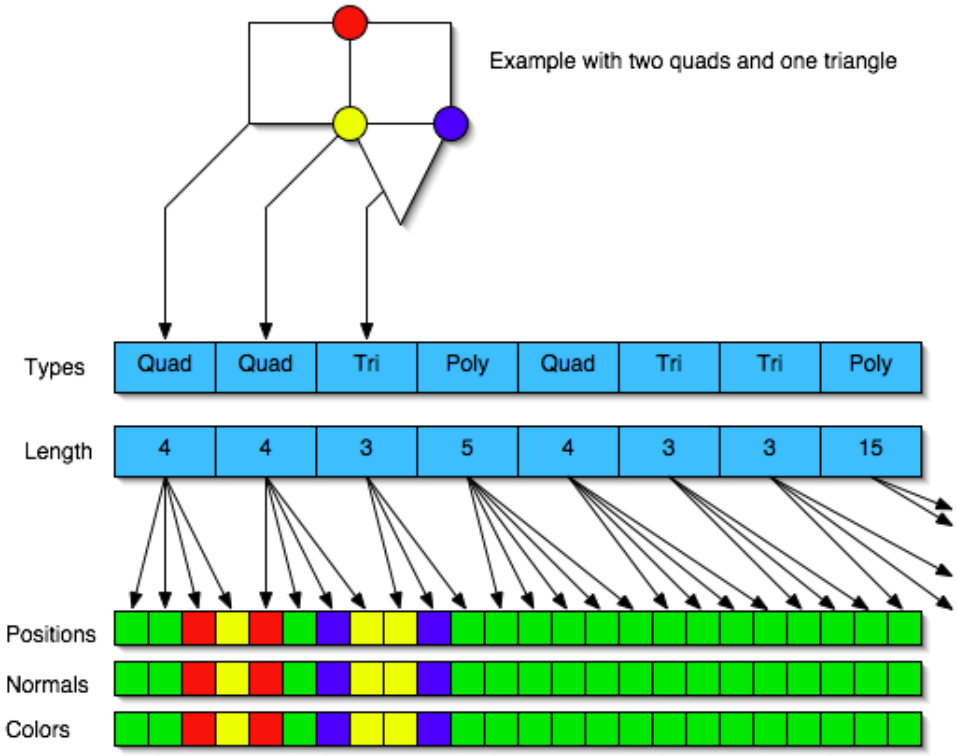
Polja podataka

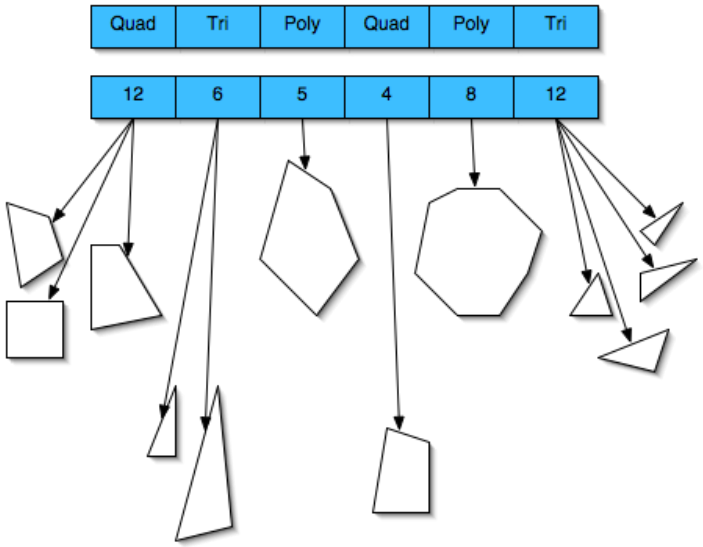
- veliki skupovi podataka i mreže poligona
  - http://threejs.org/examples/#webgl_buffergeometry
  - http://alteredqualia.com/three/examples/webgl_buffergeometry_perf2.html

- mreže poligona
  - niz poligona (grupa poligona) sa svim informacijama o poligonu
  - vrhovi i poligoni su obično pohranjeni u polja – način organizacije ovisi o daljnjim postupcima nad objektom (npr. eksplozija)
  - vrh ima niz atributa (u posebnim poljima)
    - koordinate, boja, normala, koordinate teksture → Vertex Buffer Object b
  - polje poligona
    - niz pokazivača (indeksa) na vrhove → Element Array Buffer b

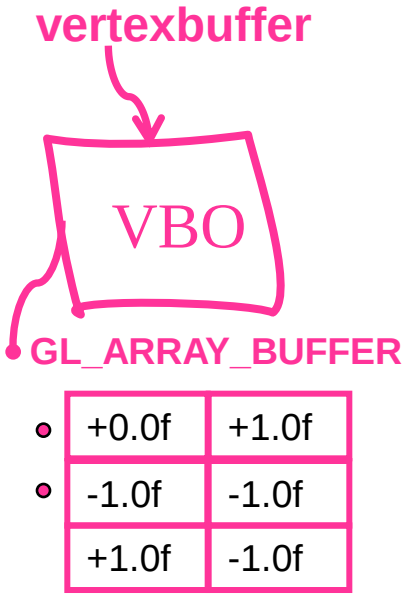
Organizacija podataka **bez indeksiranja**:

- višestruki podaci
- pogodno npr. za eksploziju poligona





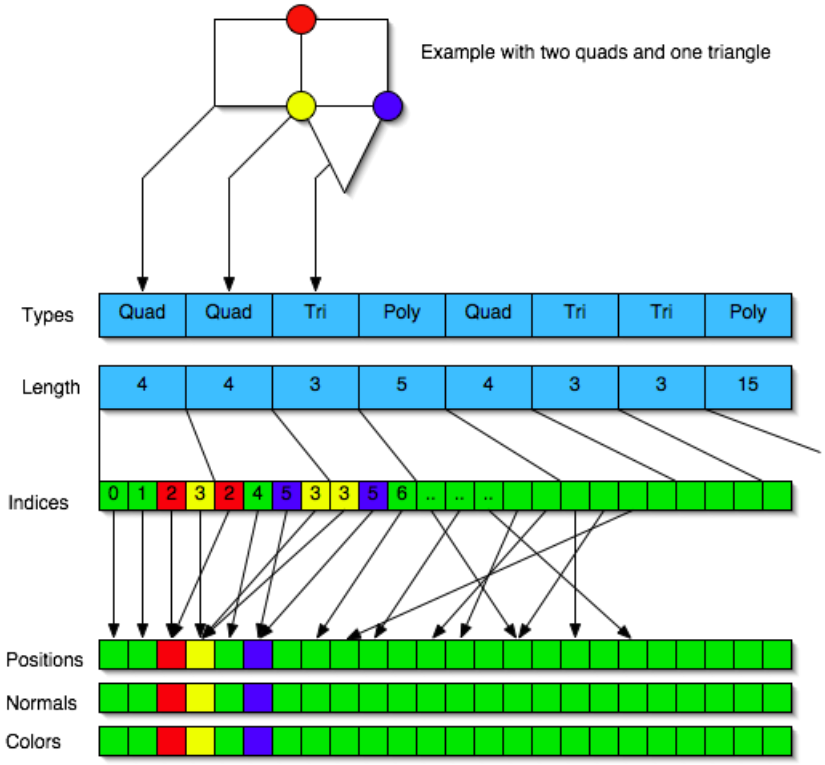
Organizacija podataka **bez indeksiranja**:



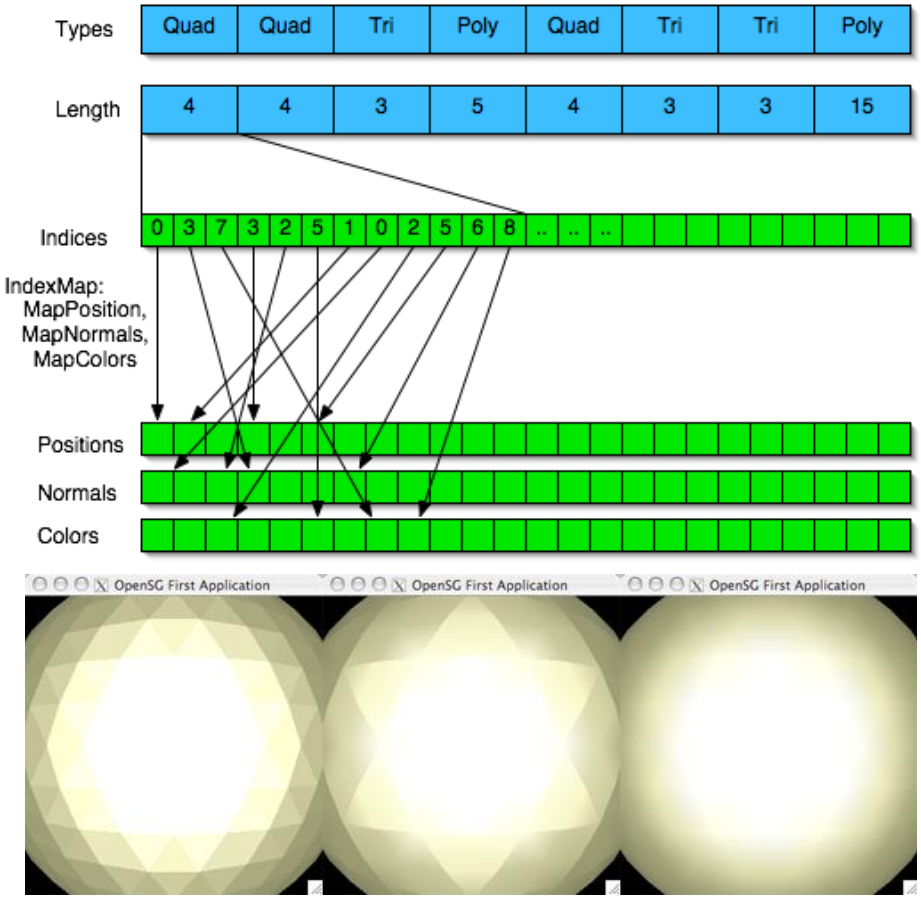

```
// VBO - VertexBufferObject

GLuint vertexbuffer;  // Logička adresa (pokazivač)
glGenBuffers(kolikoSpremnikaTrebaStvoriti, &identifikatorSpremnika)
void glBindBuffer(GLenum target, GLuint buffer);
// target - vrsta spremnika (Binding Point): GL_ARRAY_BUFFER vs.
GL_ELEMENT_ARRAY_BUFFER – indeksirani pristup
glBufferData(GL_ARRAY_BUFFER, sizeof(g_vertex_buffer_data),
g_vertex_buffer_data, GL_STATIC_DRAW);

// Moramo opisati što su podaci koje šaljemo i poslati ih u protočni sustav
void glVertexAttribPointer(GLuint index, GLint size, GLenum type,
GLboolean normalized, GLsizei stride, const GLvoid * pointer);
```

```
// VBO – VertexBufferObject

GLfloat verts[] = { +0.0f, +1.0f,    -1.0f, -1.0f,     +1.0f, -1.0f}; // Želimo prenijeti na GKarticu -> VBO
GLuint vertexbuffer;
glGenBuffers(1, &vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(verts), verts, GL_STATIC_DRAW);

glVertexAttribPointer(0,  // indeks atributa, koji mora odgovarati indeksu u sjenčaru
                      2,  // broj koordinata u vrhu – mora biti 1, 2, 3 ili 4
                      GL_FLOAT, // tip koordinate
                      GL_FALSE, // automatska normalizacija na raspon [-1, 1]
                      0,        // razmak u oktetima između vrhova; 0: slijedni vrhovi
                      (void*)0 );  // indeks okteta u nizu na kojem počinju vrhovi
glEnableVertexAttribArray(0);
```

- **Indeksirana** organizacija podataka
  - zajednički podaci za pojedini vrh
  - npr. sjenčanje Gouraud

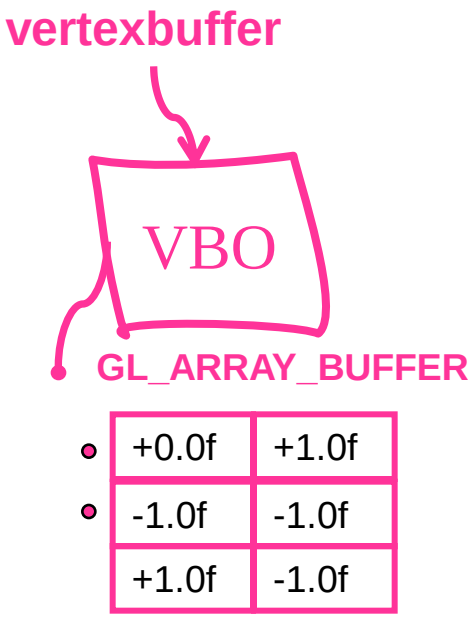


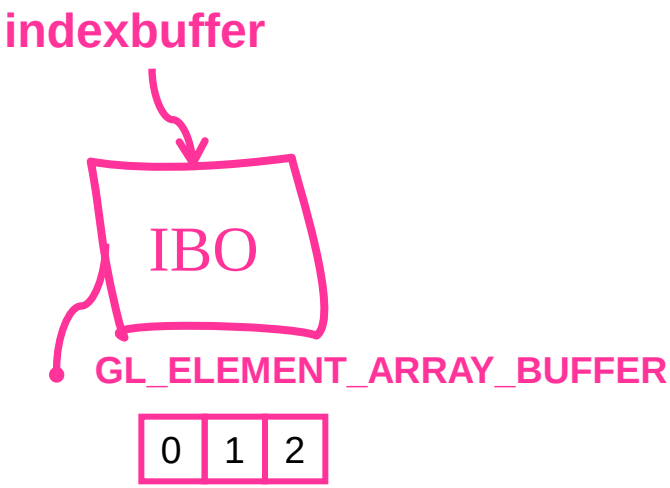
- **višestruko indeksirana** organizacija podataka
  - poseban pristup normalama, boji i sl.
  - povećan broj indeksa (×3) poligona



Indeksirana organizacija podataka

```
// Indeksirani vrhovi – GL_ELEMENT_ARRAY_BUFFER

GLushort vindices[] = {0, 1, 2};
GLuint indexbuffer;
glGenBuffers(1, &indexbuffer);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, indexbuffer);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(vindices), vindices, GL_STATIC_DRAW);
…

// glDrawArrays(GL_TRIANGLES, 0, 3); // ako je neindeksirano
glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_SHORT, 0);
```





```
// Dodavanje Atributa Boje

GLfloat verts[] = {  +0.0f, +1.0f,    +1.0f,  +0.0f, +0.0f,
                     -1.0f, -1.0f,    +0.0f,  +1.0f, +0.0f,
                     +1.0f, -1.0f ,   +0.0f,  +0.0f, +1.0f};   // Dodali smo boju

GLuint vertexbuffer;
glGenBuffers(1, &vertexbuffer);
glBindBuffer(GL_ARRAY_BUFFER, vertexbuffer);
glBufferData(GL_ARRAY_BUFFER, sizeof(verts), verts, GL_STATIC_DRAW);

glEnableVertexAttribArray(0);
glVertexAttribPointer(0,  2,  GL_FLOAT,  GL_FALSE,
                      sizeof(float) * 5,      // razmak u oktetima između vrhova,   0, // slijedni vrhovi
                      (void*)0  // indeks okteta u nizu na kojem počinju vrhovi);

glEnableVertexAttribArray(1);   // Novi atribut je boja
glVertexAttribPointer(1 // Koristi polje boje,  3 // Boja ima 3 * float ,  GL_FLOAT,  GL_FALSE,
                      sizeof(float) * 5,      // razmak u oktetima između vrhova  0, // slijedni vrhovi
                      (char*)(sizeof(float)  * 2));  // pomak do početka podataka o boji

// Potrebno je napisati sjenčare – minimalno vertex koji će proslijediti boju u fragment da bi
trokut bio obojan

glDrawElements(GL_TRIANGLES, 3, GL_UNSIGNED_SHORT, 0);
```

3.3 Sjenčari (*Shaders)*

- GLSL - OpenGL Shading Language
  - Specifikacija https://www.khronos.org/registry/OpenGL/specs/gl/GLSLangSpec.4.60.pdf
  - Moguće programiranje nekoliko faza unutar protočne strukture na grafičkoj kartici uz paralelno izvođenje
  - Sjenčar vrhova 

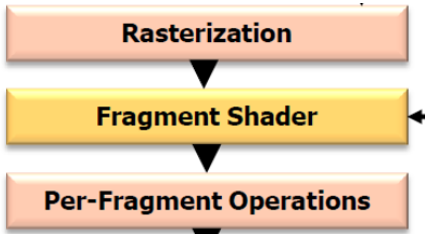
    - izvršavanje nad svakom točkom (promjena koordinata vrhova)
  - Sjenčar geometrije i sjenčar teselacije 

    - Izvršavanje nad skupom točaka (smanjivanje ili povećavanje broja točaka/poligona)
  - Sjenčar fragmenata 

    - Izvršavanje nad podacima o fragmentima koji se zapisuju u međuspremnik za prikaz slike (promjena boje, texturiranje, osvjetljenje) → 

  - Sjenčar računanja – ostvarenje GPGPU (OpenCL)
- https://www.shadertoy.com/view/XtGfRW
- https://www.shadertoy.com/view/ltyfzD
- https://www.shadertoy.com/view/Ms2SD1

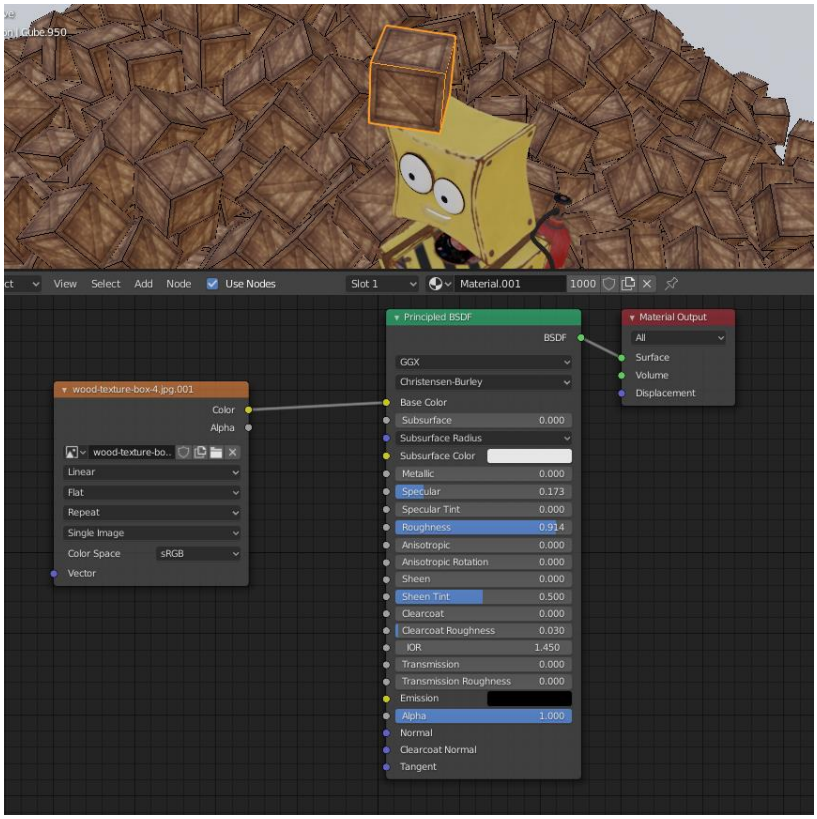
Primjer sjenčara fragmenta u GLSL




```
#version 330 core
//ulazni parametri ovisni o instanci pokrenutog sjenčara
in vec2 texCoord;
in vec4 color;
in vec3 normal;
in vec3 fragPos;

//ulazni parametri jednaki svakom pokrenutom sjenčaru fragmenata
uniform sampler2D tex;
uniform vec3 lightPos;

void main() {
  gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0); //svaki fragment crvene boje
  //boja fragmenta ovisna o interpoliranoj vrijednosti između 3 vrha trokuta
  gl_FragColor =color;
  //boja objekta s teksturom * (ambijentalna komponenta + difuzna komponenta)
  gl_FragColor = texture(tex, texCoord) *
                 (0.4 + dot(normal, normalize(lightPos - fragPos)));
}
```

Graf sjenčara

- moguća je reprezentacija sjenčara u obliku grafa
- pogodnije za jednostavne i učestale sjenčare te za eksperimentiranje s idejama
- Česte komponente dostupne kroz biblioteke, a moguća izrada i vlastitih čvorova
- Blender, Unity, Unreal Engine..



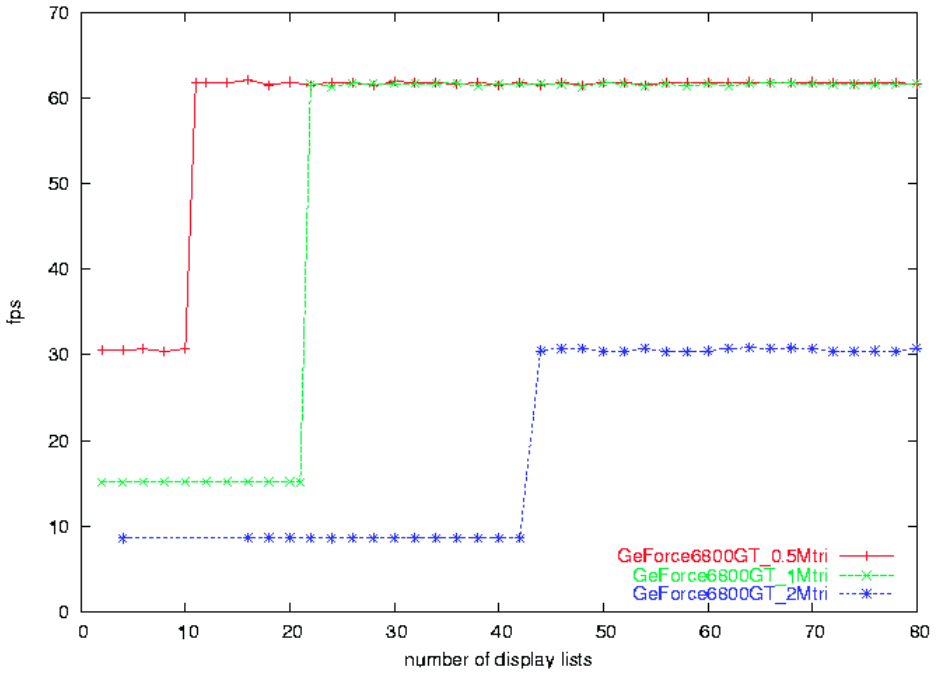
3.4 Performanse aplikacije

Određivanje “uskog grla” (bottlenecks)

- izvori problema
  - CPU – prespor – smanjiti količinu obrade na CPU
  - širina sabirnice - propusnost
  - veličina memorije (Cache size, RAM)
  - količina podataka o geometriji – GPU
  - Stapanje geometrijskih podataka – „promet” CPU – GPU (60x)
  - https://threejs.org/examples/?q=instancing#webgl_instancing_performance
  - http://www.visualiser.fr/Babylon/character/default.htm
- efikasne strukture podataka
- **binary** ili ascii spremanje podataka u datoteku
- Uloženo vrijeme programera

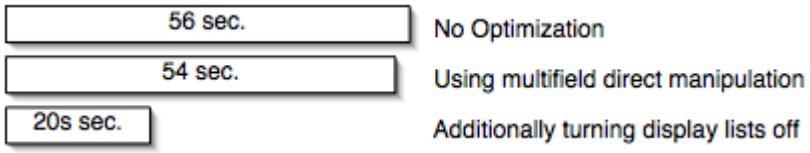
Primjer problema kod prikaznih listi

- Umjesto pojedinačnog prenošenja svake naredbe grafičkoj kartici može ih se više zapakirati u prikaznu listu te ih odjednom izvršiti na grafičkoj kartici
- Povećanjem broja listi smanjuje se količina naredbi/podataka u pojedinoj listi – kada jedna lista stane u priručnu memoriju ubrza se izvođenje (s 15 na 60 FPS inače swap-a)
  - 0.5 M tri - 10 lista
  - 1 M tri – 21 lista
  - 2 M tri – 43 liste

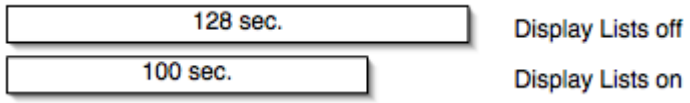


Primjer problema kod prikaznih listi

- Korištenje *Static Objects* vs. *Dynamic objects*
  - potrebno je određeno dodatno vrijeme (overhead) pri izgradnji prikaznih listi
  - prikazne liste imaju smisla kod statičkih objekata - sada imamo vertex shadere pa objekti mogu biti dinamički (mogu mijenjati oblik u vremenu)
  - programi za sjenčanje geometrije – promjena broja poligona
  - OpenSceneGraph koristi prikazne liste (default) pa je kod dinamičkih objekata potrebno isključiti njihovo korištenje



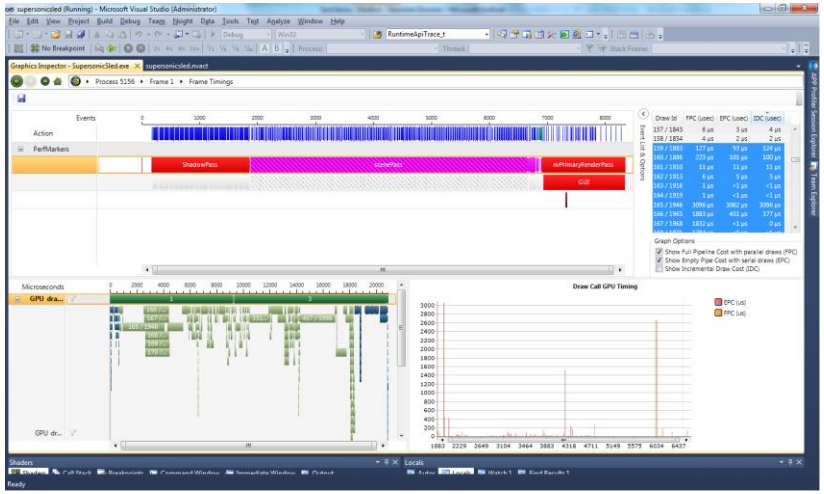
Display Lists on Dynamic objects
(npr. površina vode)



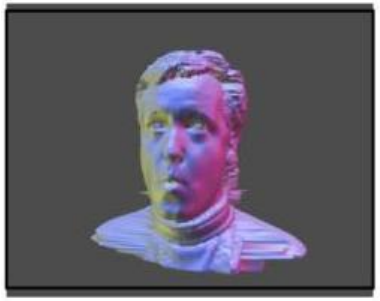
Display Lists on Static Objects

Mjerenje performansi grafičke aplikacije

- u analizi nas zanima broj funkcijskih poziva i vrijeme potrošeno unutar pojedinog poziva OpenGL naredbi (programa za sjenčanje npr. GLSL) – **profiler**
- PerfHUD (PerfKit) alati za analizu Direct3D aplikacija
- NVIDIA® Nsight http://www.nvidia.com/object/nsight.html
- analiza - statistike http://www.babylonjs.com/Demos/InstancedBones/
https://spector.babylonjs.com/demos/instancedbones/#



Scenes: Head (240 frames)



Vertices	60104
Triangles (3D)	59592
Triangles (2D)	24884
Fragments	263369
Image	1024×768

60104	glNormal3fv
60104	glVertex3fv
803	glVertex2fv
722	glLoadMatrixf
720	glMultMatrixf
481	glColor3fv
452	glBegin
452	glEnd
241	glClear
240	gltSwapBuffers
240	glCallList
220	glEndList
220	glNewList
124	glTranslatef
19	glLightfv
9	glEnable
6	glPixelStorei
2	glClearColor
2	glDrawBuffer
2	glLightModelf
2	glMaterialfv
2	glMatrixMode
2	glViewport
1	gltPad

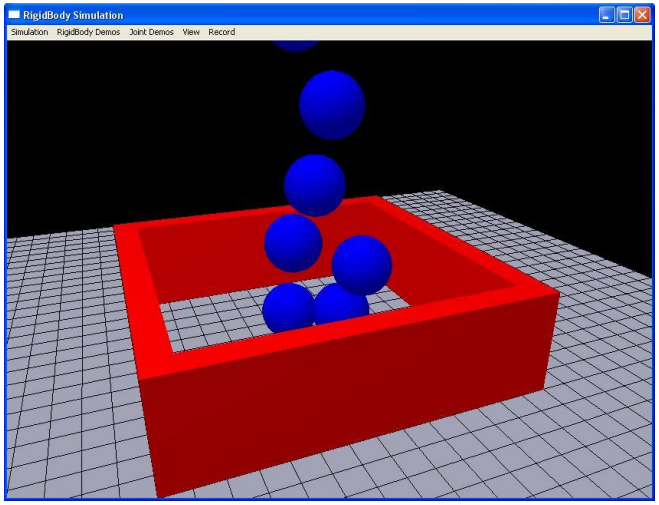
CS448 Lecture 4

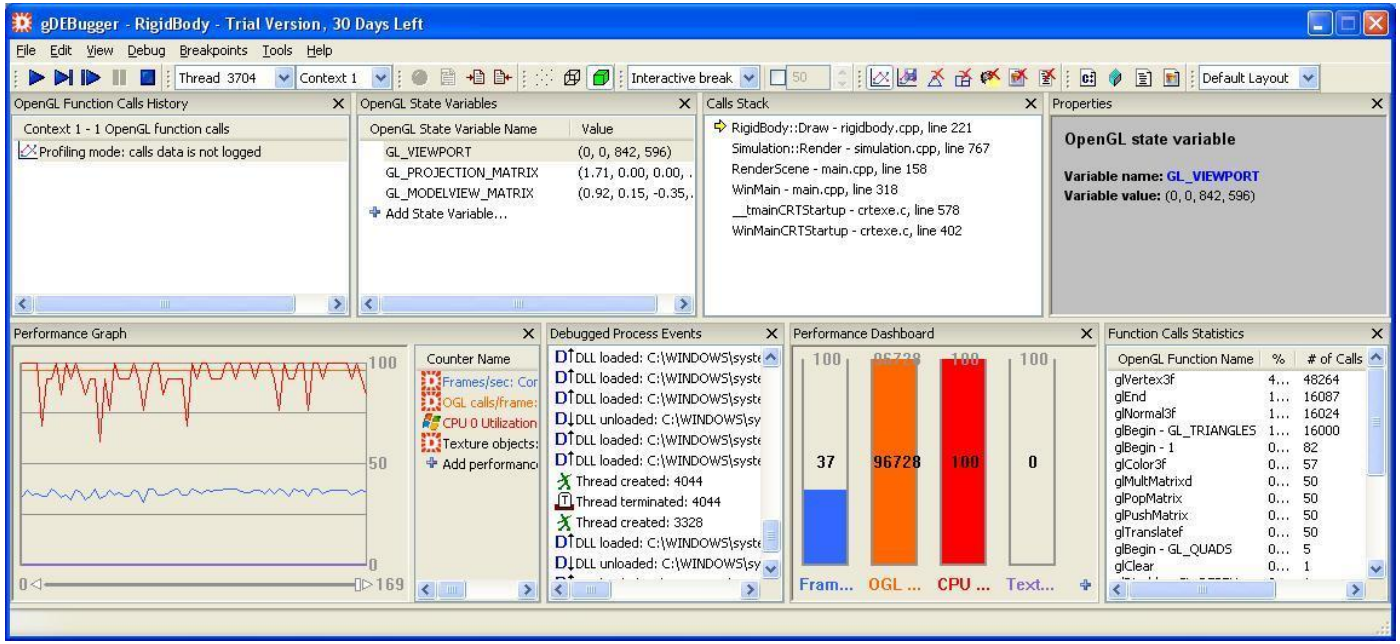
Kurt Akeley, Pat Hanrahan, Fall 2001

Npr. aplikacije za analizu grafičkih programa

- gDEBugger
  - analiza funkcijskih poziva,

  varijabli stanja, CPU
  - memorija L1, L2
- RenderDoc https://renderdoc.org/
- WebGL-Inspector
  - http://benvanik.github.io/WebGL-Inspector/
  - http://www.realtimerendering.com/blog/webgl-debugging-and-profiling-tools/
- Primjer izvođenja aplikacije
  - https://threejs.org/examples/#webgl_animation_scene Web Devel/Perf





Mjerenje performansi grafičkih aplikacija

SPEC Standard Performance Evaluation Corporation

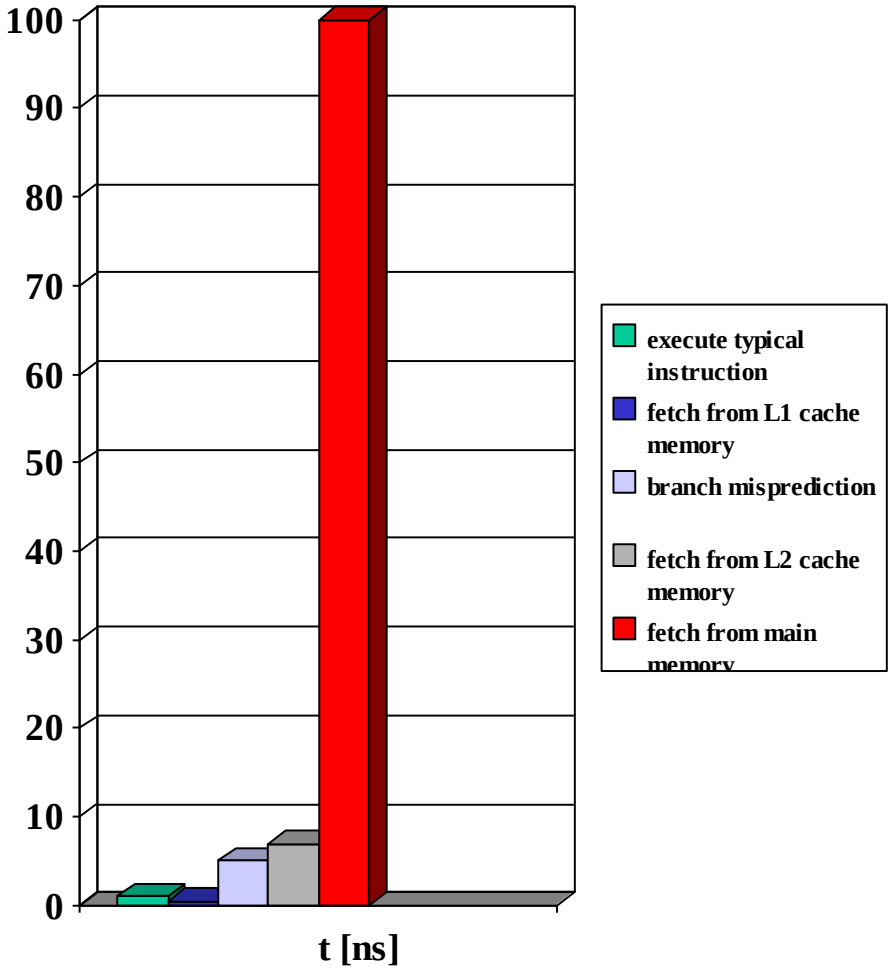
- GPC – Graphics Performance Characterization
- SPECviewperf®
- http://www.specbench.org/gpc/downloadindex.html
- http://www.spec.org/gwpg/
  - 3ds max™ 6
  - Maya® 5
  - Pro/ENGINEER™
  - Solid Edge V14
  - SolidWorks 2003™
- mjerenje uz uključivanje raznih opcija
  http://www.specbench.org/gpc/opc.static/whatis_vp8.html

Futuremark - PCMark 10, 3DMark, VRMark http://www.futuremark.com/

- PCMark - mjerenje vezano uz konfiguraciju računala
- 3DMark – računalne igre
- VRMark – performanse virtualnog okruženja

Vremena izvođenja raznih operacija na tipičnom PC:

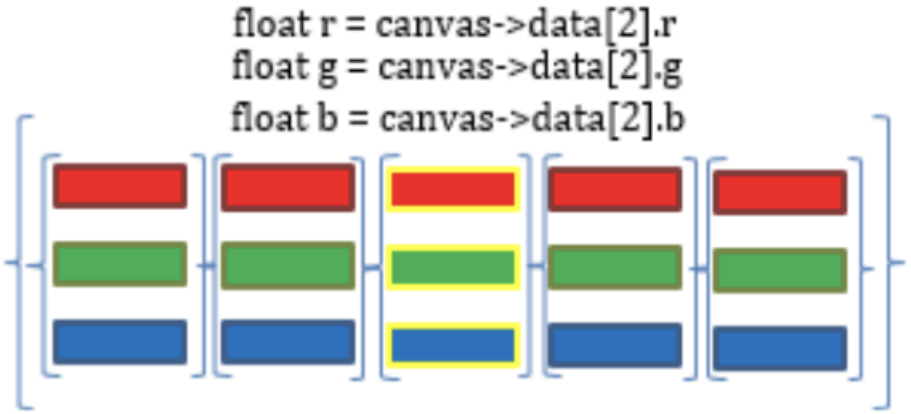
	t/ ns
Izvršavanje CPU instrukcije	1
Dohvat podataka iz L1 priručne memorije (engl. cache)	0,5
Krivo predviđanje grananja	5
Dohvat podataka iz L2 priručne memorije	7
Dohvat podataka iz glavne memorije	100
Slanje 2KB preko 1Gbps mreže	20 000
Sekvencionalno čitanje 1MB iz glavne memorije	250 000
Dohvaćanje podataka s nove pozicije (seek)	8 000 000
Sekvencionalno čitanje 1MB sa SSD	2 000 000
Sekvencionalno čitanje 1MB sa HDD	20 000 000
Slanje paketa iz US u Europu i nazad	150 000 000



Pohrana podataka u memoriji, primjer za pohranu slike (boje):

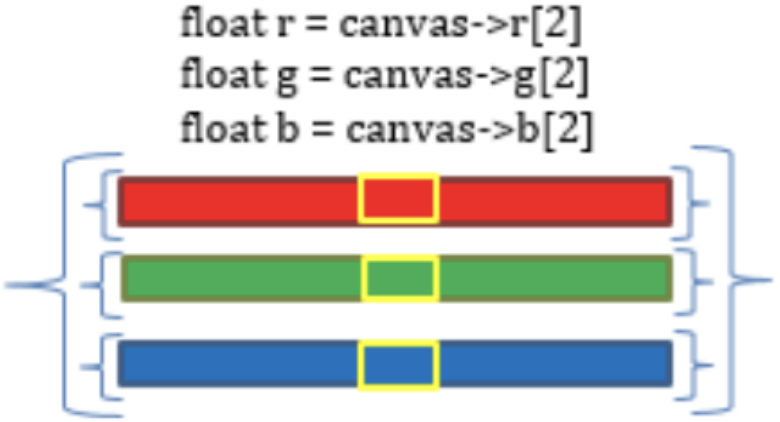
Array-of-Structs (AoS)

polje struktura RGBA
- organizirano po pikselu:



Struct-of-Arrays (SoA)

struktura polja
- organizirano po kanalu R G B A



Npr: C++

AoS - podaci o pojedinom elementu će biti brže dohvatljivi u priručnoj memoriji (Cache) pa je u većini slučajeva AoS bolje

SoA – ako radimo sekvencijsku obradu same slike (podataka) tada je bolja SoA

Polje struktura (AoS)

```
struct RGBA{
    uint8_t red;
    uint8_t green;
    uint8_t blue;
    uint8_t alpha;
}

std::vector<RGBA> canvas;
```

Struktura polja (SoA)

```
struct Canvas{
    std::vector<uint8_t> redChannel;
    std::vector<uint8_t> greenChannel;
    std::vector<uint8_t> blueChannel;
}

Canvas canvas;
```