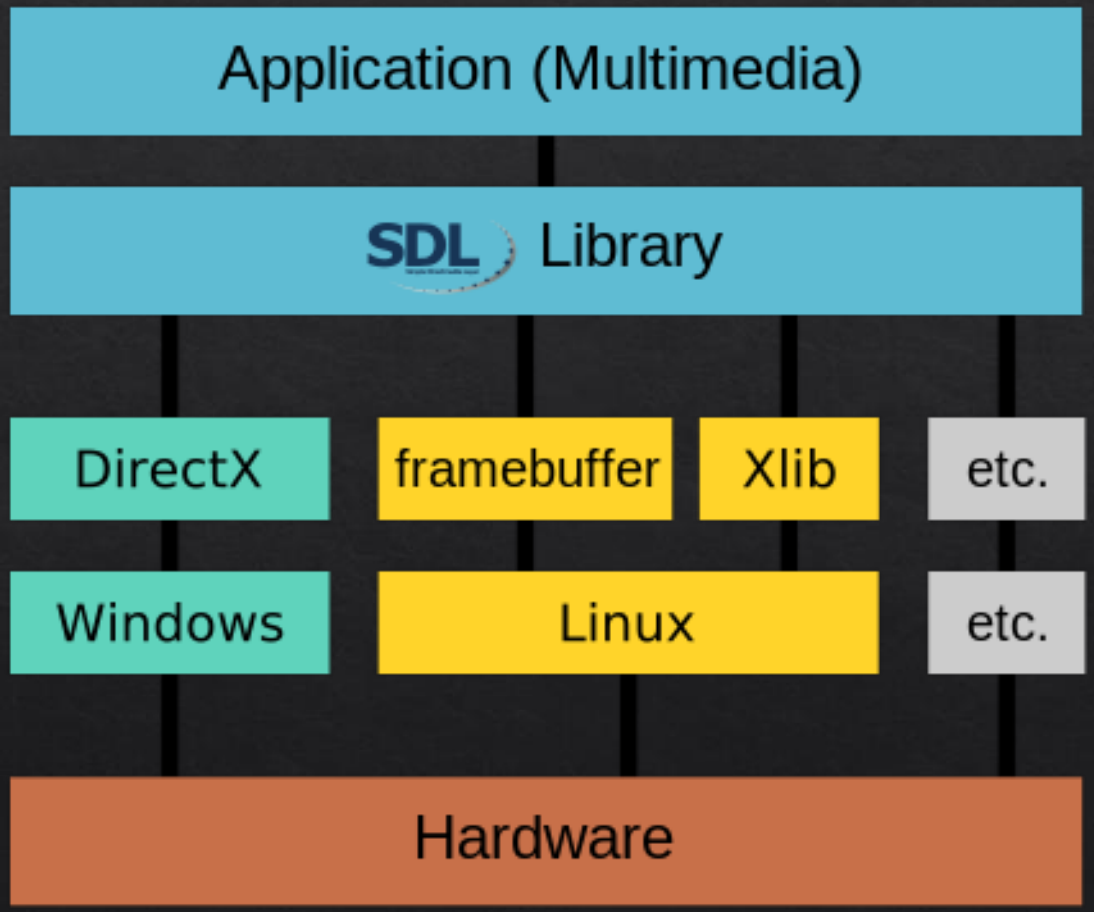

GAME PROGRAMMING IN C++
2020

Biblioteke za razvoj igara

- Na našu sreću, postoje ljudi koji su i taj korak napravili umesto nas
- https://github.com/Calinou/awesome-gamedev#programming-frameworks-and-libraries
- SDL, SFML, pyGame, LÖVE, GLFW, Three.js, bgfx
- I dalje zahtevaju dosta programiranja, ne treba ih pomešati sa Unity i Unreal-om

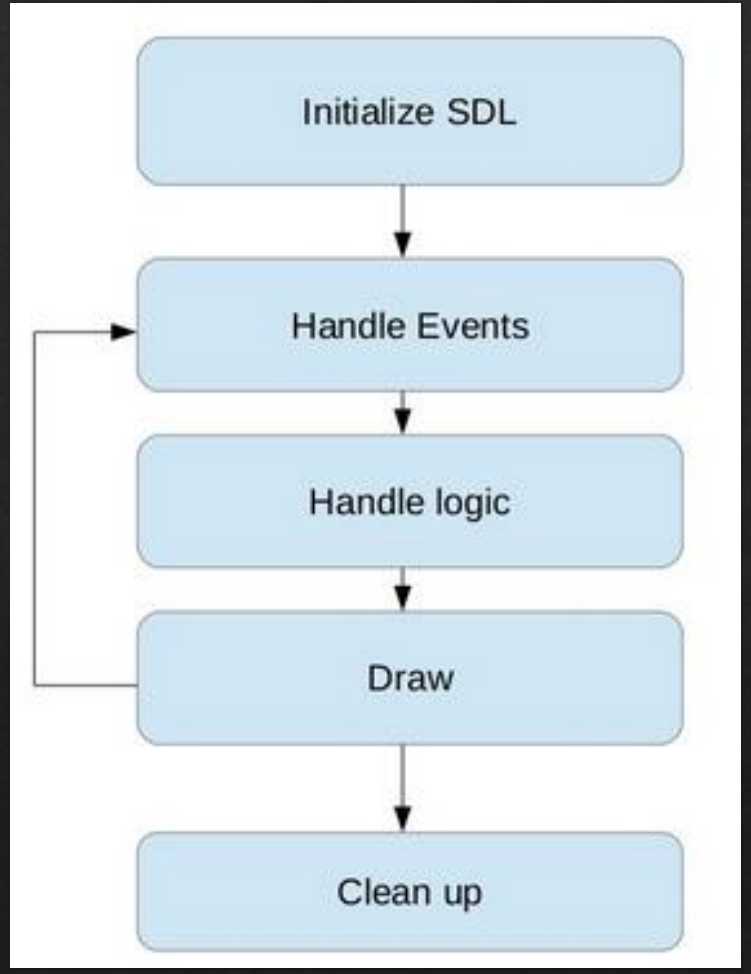
SDL2


Application (Multimedia)
SDL Library
DirectX
framebuffer
Xlib
etc.
Windows
Linux
etc.
Hardware

SDL2 podsistemi

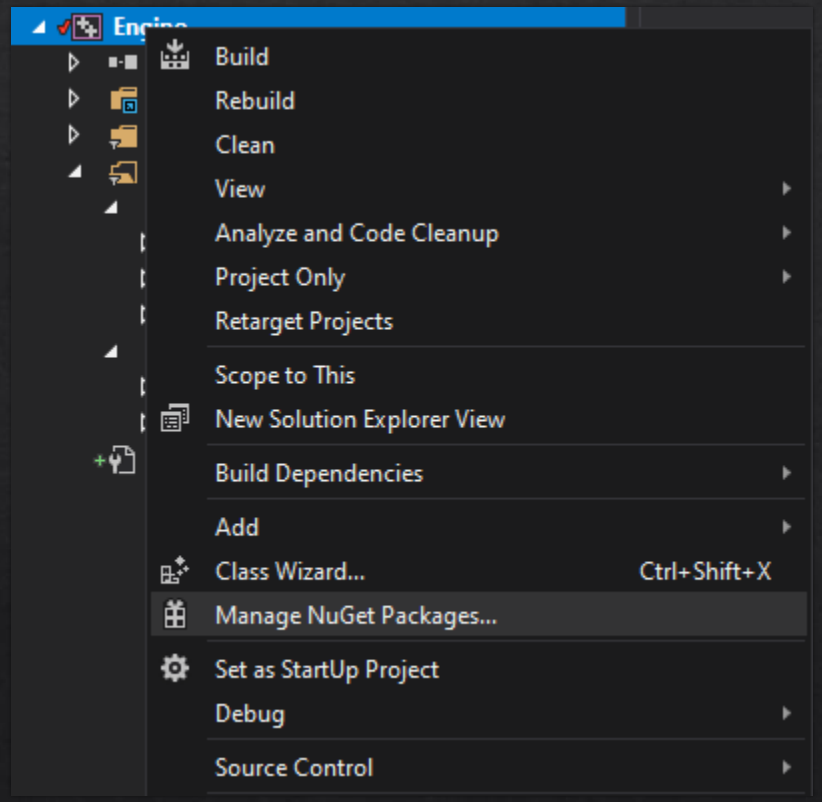
Basics	Initialization and Shutdown, Configuration Variables, Error Handling, Log Handling
Video	Display and Window Management, surface functions, rendering acceleration, etc.
Input Events	Event handling, Support for Keyboard, Mouse, Joystick and Game controller
Audio	SDL_audio.h implements Audio Device Management, Playing and Recording
Threads	multi-threading: Thread Management, Thread Synchronization Primitives, Atomic OperationsFile Abstraction

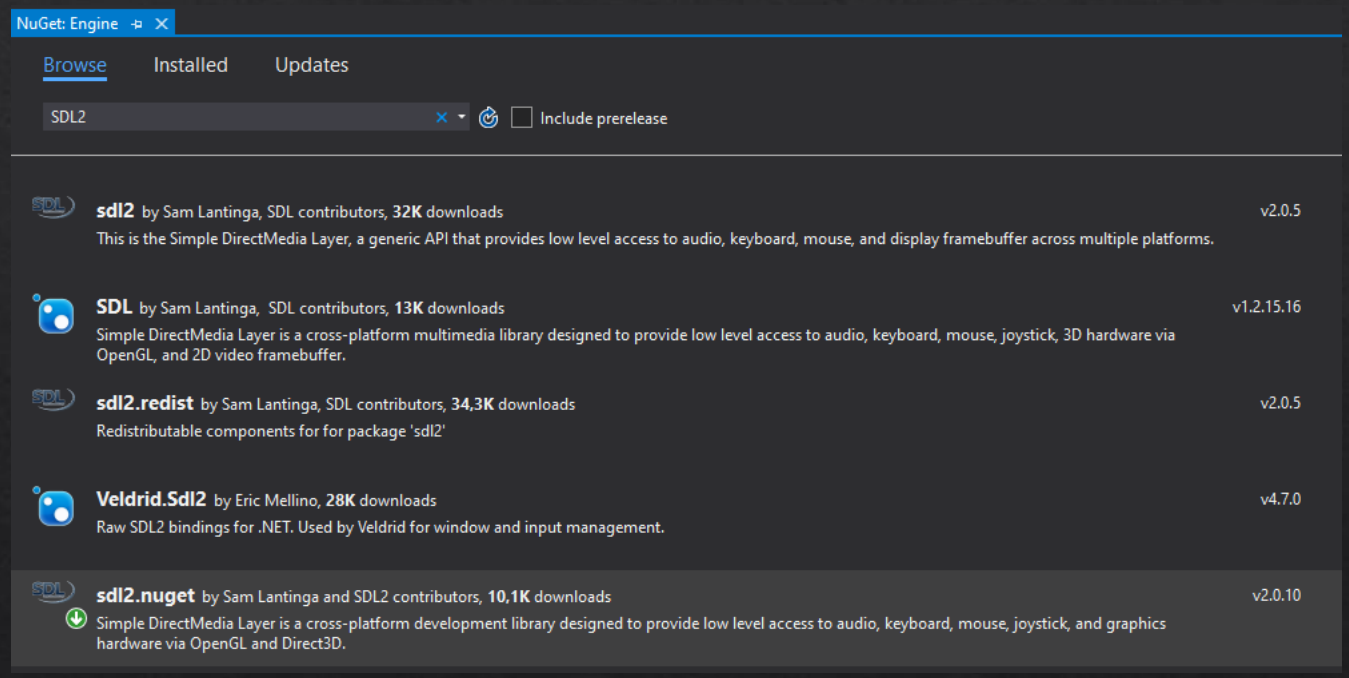
Struktura SDL aplikacija



https://www.slideshare.net/EricBasile/sdl2-gamedevelopment-vt-code-camp

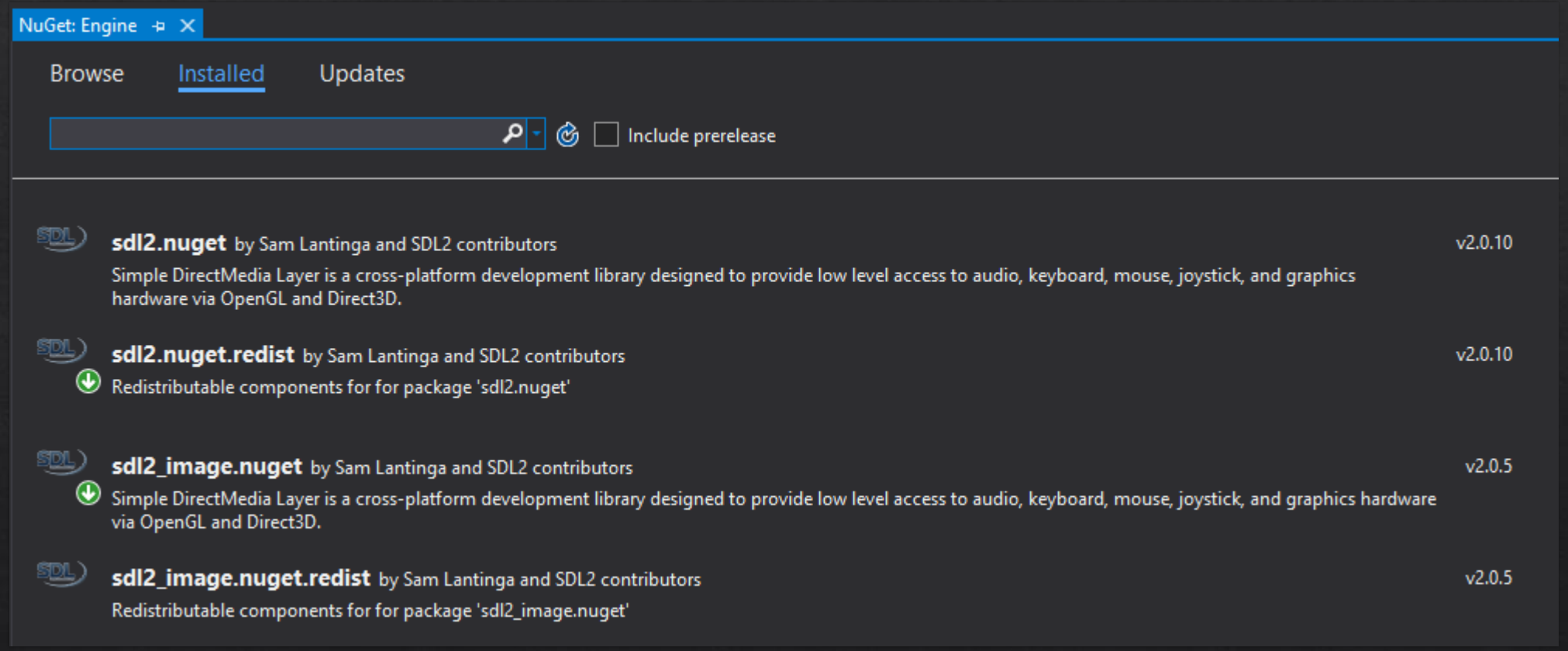
Dodavanje SDL2 u projekat



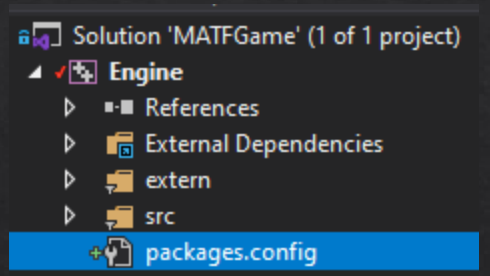


isto i za sdl2_image.nuget

Dodavanje SDL2 u projekat



Dodavanje SDL2 u projekat



packages.config

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="sdl2.nuget" version="2.0.10" targetFramework="native" />
  <package id="sdl2.nuget.redist" version="2.0.10" targetFramework="native" />
  <package id="sdl2_image.nuget" version="2.0.5" targetFramework="native" />
  <package id="sdl2_image.nuget.redist" version="2.0.5" targetFramework="native" />
</packages>
```

Korišćenje SDL2

- Mi ćemo koristiti samo za render
- http://lazyfoo.net/tutorials/SDL/
- Pošto je C biblioteka, ne možemo da koristimo RAII

Korišćenje SDL2

Use this function to initialize the SDL library. This must be called before using most other SDL functions.

```
int SDL_Init(Uint32 flags)
```

SDL_INIT_TIMER

SDL_INIT_AUDIO

SDL_INIT_VIDEO

SDL_INIT_JOYSTICK

SDL_INIT_HAPTIC

SDL_INIT_GAMECONTROLLER

SDL_INIT_EVENTS

SDL_INIT_EVERYTHING

SDL_INIT_NOPARACHUTE

```c
#include "SDL.h"

int main(int argc, char* argv[])
{
    if (SDL_Init(SDL_INIT_VIDEO|SDL_INIT_AUDIO) != 0) {
        SDL_Log("Unable to initialize SDL: %s", SDL_GetError());
        return 1;
    }

    /* ... */

    SDL_Quit();

    return 0;
}
```

https://wiki.libsdl.org/

Korišćenje SDL2

Use this function to create a window with the specified position, dimensions, and flags.

```
SDL_Window* SDL_CreateWindow(const char* title,
                             int         x,
                             int         y,
                             int         w,
                             int         h,
                             Uint32      flags)
```

title the title of the window, in UTF-8 encoding

x the x position of the window, SDL_WINDOWPOS_CENTERED, or SDL_WINDOWPOS_UNDEFINED

y the y position of the window, SDL_WINDOWPOS_CENTERED, or SDL_WINDOWPOS_UNDEFINED

w the width of the window, in screen coordinates

h the height of the window, in screen coordinates

flags 0, or one or more SDL_WindowFlags OR'd together; see Remarks for details

```
int main(int argc, char* argv[]) {

    SDL_Window *window;                    // Declare a pointer

    SDL_Init(SDL_INIT_VIDEO);              // Initialize SDL2

    // Create an application window with the following settings:
    window = SDL_CreateWindow(
        "An SDL2 window",                  // window title
        SDL_WINDOWPOS_UNDEFINED,           // initial x position
        SDL_WINDOWPOS_UNDEFINED,           // initial y position
        640,                               // width, in pixels
        480,                               // height, in pixels
        SDL_WINDOW_OPENGL                  // flags - see below
    );
```

https://wiki.libsdl.org/

Korišćenje SDL2

Use this function to create a 2D rendering context for a window.

```
SDL_Renderer* SDL_CreateRenderer(SDL_Window* window,
                                 int         index,
                                 Uint32      flags)
```

window the window where rendering is displayed

index the index of the rendering driver to initialize, or -1 to initialize the first one supporting the requested flags

flags 0, or one or more SDL_RendererFlags OR'd together; see Remarks for details

```
 SDL_Init(SDL_INIT_VIDEO);

 win = SDL_CreateWindow("Hello World", posX, posY, width, height, 0);

 renderer = SDL_CreateRenderer(win, -1, SDL_RENDERER_ACCELERATED);

SDL_RenderClear(renderer);
SDL_RenderCopy(renderer, bitmapTex, NULL, NULL);
SDL_RenderPresent(renderer);
```

https://wiki.libsdl.org/

Korišćenje SDL2

Use this function to copy a portion of the texture to the current rendering target.

```
int SDL_RenderCopy(SDL_Renderer*   renderer,
                   SDL_Texture*    texture,
                   const SDL_Rect* srcrect,
                   const SDL_Rect* dstrect)
```

renderer	the rendering context
texture	the source texture; see Remarks for details
srcrect	the source SDL_Rect structure or NULL for the entire texture
dstrect	the destination SDL_Rect structure or NULL for the entire rendering target; the texture will be stretched to fill the given rectangle

```
SrcR.x = 0;
SrcR.y = 0;
SrcR.w = SHAPE_SIZE;
SrcR.h = SHAPE_SIZE;

DestR.x = 640 / 2 - SHAPE_SIZE / 2;
DestR.y = 580 / 2 - SHAPE_SIZE / 2;
DestR.w = SHAPE_SIZE;
DestR.h = SHAPE_SIZE;
SDL_RenderCopy(Main_Renderer, BlueShapes, &SrcR, &DestR);
SDL_RenderPresent(Main_Renderer);
```

Use this function to update the screen with any rendering performed since the previous call.

https://wiki.libsdl.org/

Naš RenderSystem

- Init()
  - Pravi SDL_Window
  - Pravi SDL_Renderer
- Update()
  - Obriše sve sa ekrana
  - Poziva SDL_Renderer za iscrtavanje svake slike u framebuffer
  - Prikazuje framebuffer na ekran
- Shutdown()
  - Uništava SDL_Renderer i SDL_Window

Naš RenderSystem

- Init()
  - Pravi Engine::Window – Wrapper oko SDL_Window
  - Pravi Engine::Renderer – Wrapper oko SDL_Renderer
- Update()
  - Obriše sve sa ekrana
  - Poziva Engine::Renderer za iscrtavanje **svake slike** u framebuffer
  - Prikazuje framebuffer na ekran
- Shutdown()
  - Uništava Engine::Renderer i Engine::Window

Naš RenderSystem

- Slike!
- Engine::Texture -> wrapper oko SDL_Texture
- Učitavaju se direktno sa diska pomoću SDL_Image pomoćne biblioteke i IMG_LoadTexture()

```
void Texture::LoadTexture(Renderer* renderer_, std::string path_)
{
    m_Texture = IMG_LoadTexture(renderer_->GetNativeRenderer(), path_.c_str());

    if (m_Texture == nullptr)
    {
        LOG_ERROR("Unable to load texture: {}, SDL_Image returned error {}", path_, IMG_GetError());
    }
}
```

Naš RenderSystem

```
namespace Engine
{
    struct WindowData;
    class Renderer;

    class RenderSystem
    {
    public:
        bool Init(const WindowData& windowData_ = WindowData());
        bool Shutdown();
        void Update(float dt_/*, EntityManager* entityManager*/);
    private:
        std::unique_ptr<Renderer> m_Renderer;
    };
}
```

Naš RenderSystem

```cpp
namespace Engine
{
    struct WindowData;
    class Window;

    class Renderer
    {
    public:
        bool Init(const WindowData& windowData_ = WindowData());
        bool Shutdown();

        void DrawImage(/* Some image parameters IDK */);
        void BeginScene() const;
        void EndScene() const;

        SDL_Renderer* GetNativeRenderer() const { return m_NativeRenderer; }

        void SetBackgroundColor(unsigned char bgR_, unsigned char bgG_, unsigned char bgB_, unsigned char bgA_);
        ~Renderer();

    private:
        std::unique_ptr<Window> m_Window;
        SDL_Renderer* m_NativeRenderer{ };

        Texture m_helloWorldTexture; // TODO: Remove after testing
    };
}
```

Naš RenderSystem

```cpp
namespace Engine {

    struct WindowData;

    class Window
    {
    public:
        bool Init(const WindowData& windowData_ = WindowData());
        bool Shutdown();
        SDL_Window* GetNativeWindowHandle() const { return m_NativeWindowHandle; }
        ~Window();
    private:
        WindowData m_WindowData{};
        SDL_Window* m_NativeWindowHandle{ };
    };
}
```

```cpp
namespace Engine {

    class Renderer;

    struct Texture
    {
        SDL_Texture* m_Texture{};

        void LoadTexture(Renderer* renderer_, std::string path_);
        ~Texture();
    };
}
```

Zadatak

- Engine::Window
  - Napraviti SDL_Window
- Engine::Render
  - Napraviti Engine::Window, napraviti SDL_Renderer
  - Učitati jednu sliku (hello_world.jpg u root-u repozitorijuma)
  - Prikazati je na ekranu (Koristeći SDL_RenderCopy)
- Pomoć
  - http://lazyfoo.net/tutorials/SDL/02_getting_an_image_on_the_screen/index.php
  - https://wiki.libsdl.org/SDL_Init

A sad teorija

Hvala na strpljenju

Rendering

- Our ability to draw graphics on the screen – TheCherno
- Rendering or image synthesis is the automatic process of generating a photorealistic or non-photorealistic image from a 2D or 3D model or scene by means of computer programs – Wikipedia

- Može biti:
  - Offline / Pre-rendered - filmovi
  - Real-time i interactive - u igrama

- Dešava se na zasebnom čipu – GPU

Rendering

- Algoritam za rendering scene

```
For each pixel on the canvas
    Paint it with the right color
```

- Malo detaljnije

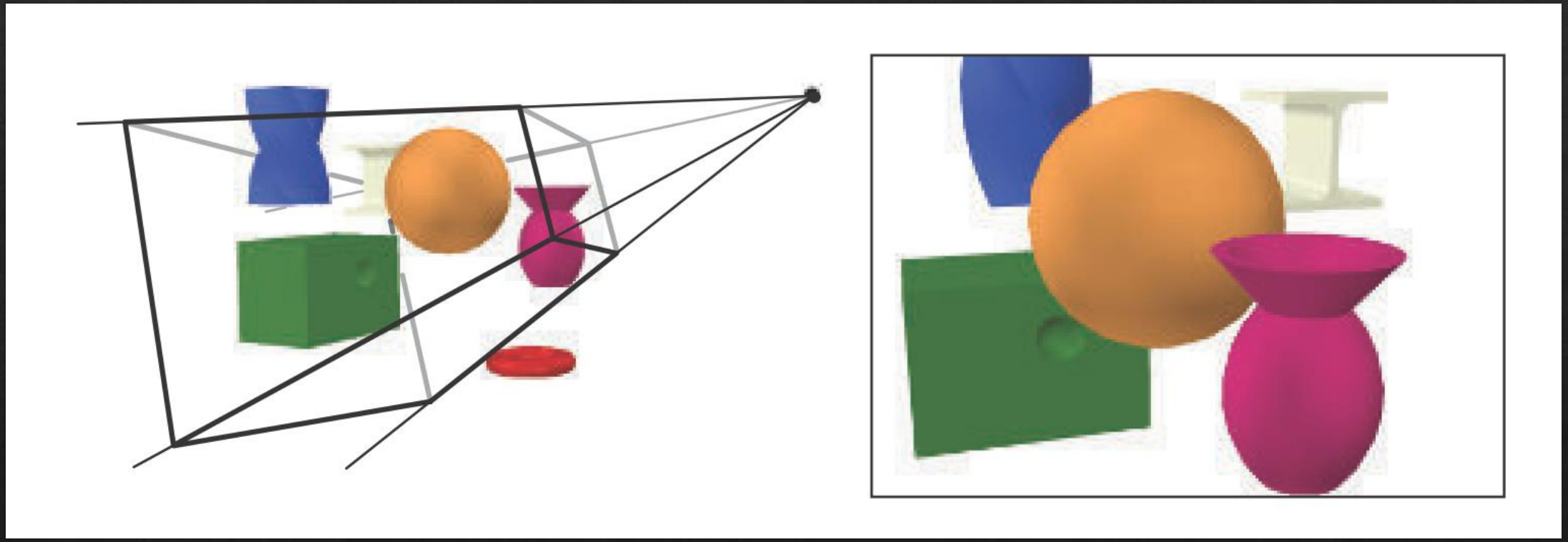
```
Place the eye and the frame as desired
For each pixel on the canvas
    Determine the square on the grid corresponding to this pixel
    Determine the color seen through that square
    Paint the pixel with that color
```

Osnove

- Scena
  - Skup 3D oblika prisutnih u prostoru
- Kamera
  - Postavljena tako da se scena hvata iz željenog ugla i udaljenosti
- Izvori svetla
  - Osvetljavaju prostor i objekte i čiji virtuelni fotoni na kraju osvetljavaju piksele kamere

Jason Gregory. 2014. *Game Engine Architecture*

Osnove



Tomas Akenine-Moller, Tomas Moller, and Eric Haines. 2002. *Real-Time Rendering* (2nd. ed.)

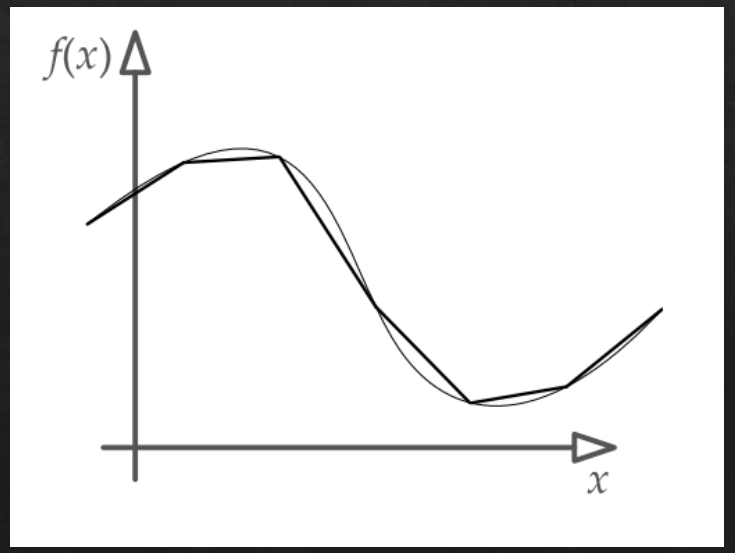
Osnove

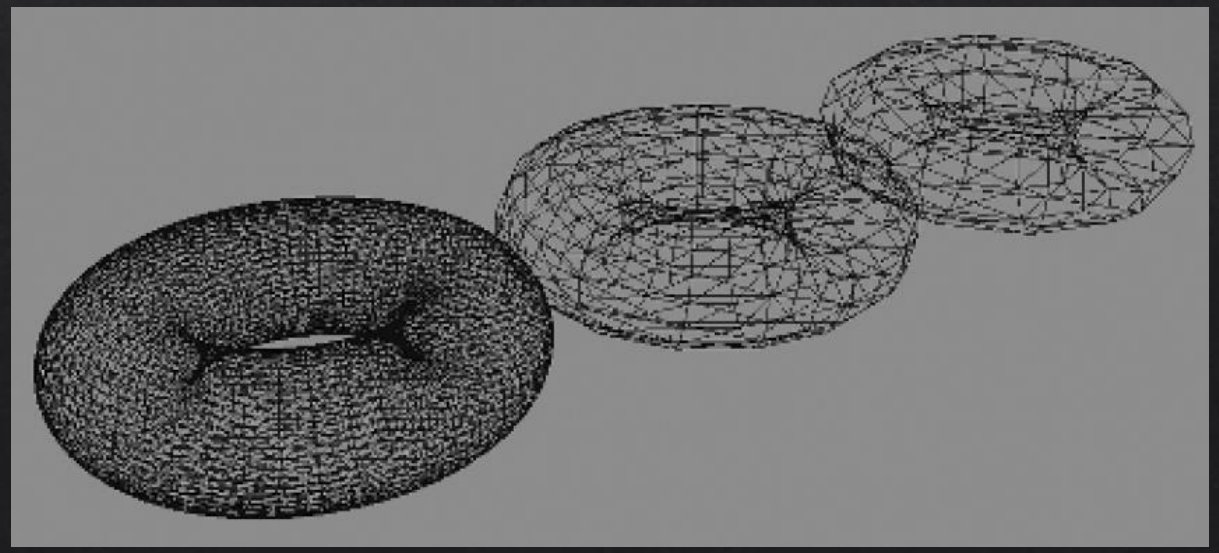
- Vizuelna svojstva 3D oblika prisutnih na sceni
- Shader
  - Programi koji se izvršavaju na GPU
  - Za svaki piksel kamere, izračunavaju se boja i intenzitet svetlosnih zraka koji konvergiraju u tu tačku
- I sve ovo 30/60/144 puta po sekundi

Jason Gregory. 2014. *Game Engine Architecture*

Oblici

- U stvari su površi predstavljene parametarskim jednačinama u 3D prostoru
- U video igrama se uglavnom predstavljaju aproksimacijama sastavljenih od trouglova – triangle mesh

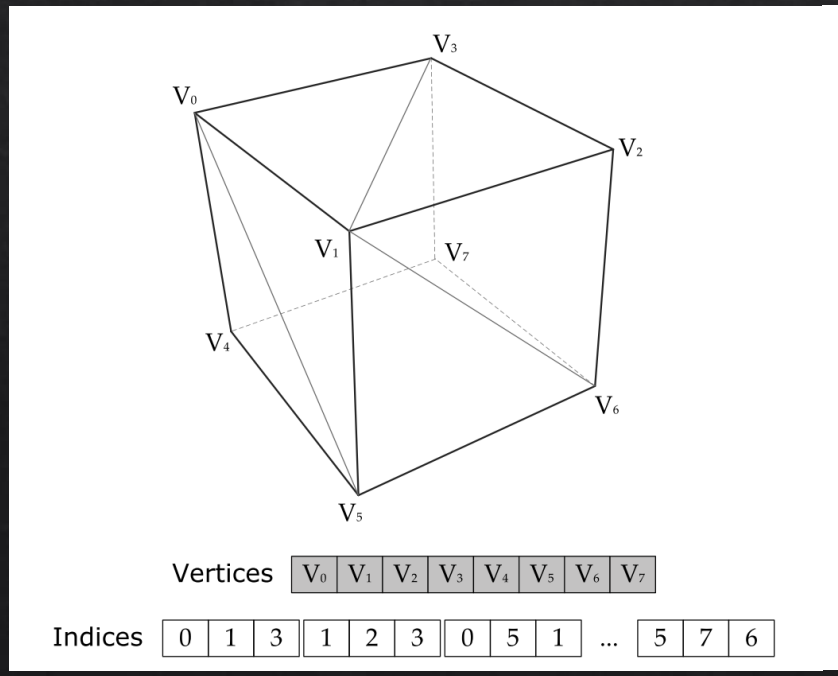


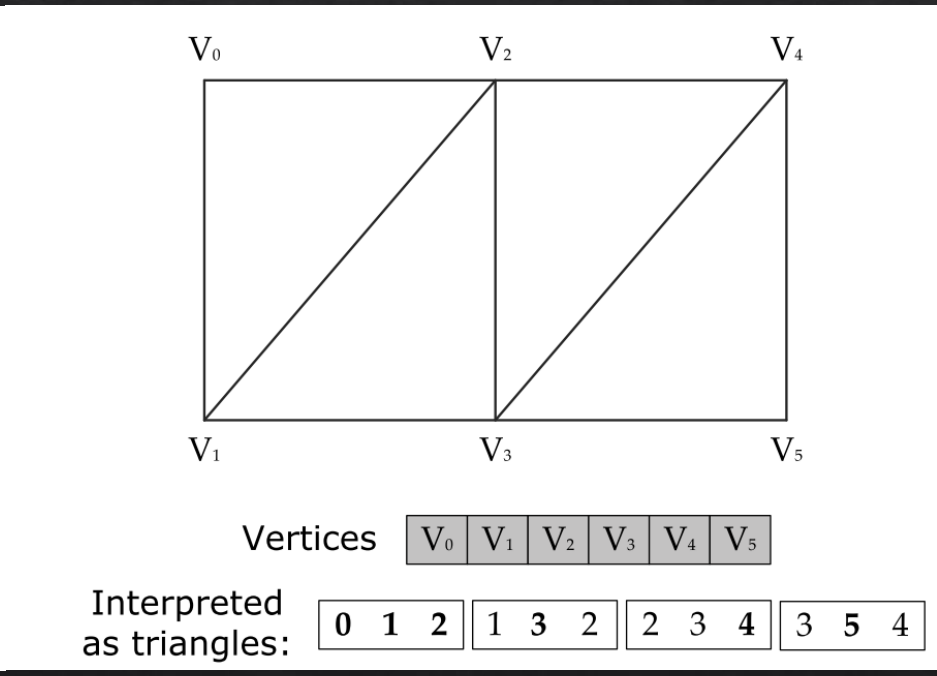


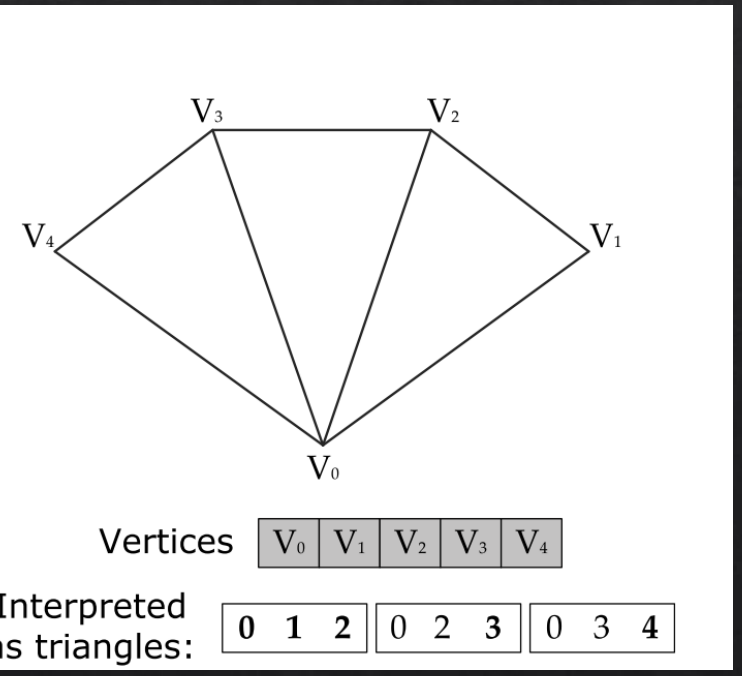
Jason Gregory. 2014. *Game Engine Architecture*

Trouglovi

- Definisani su pozicijom svojih temena (vertices) i normalom
- Svako teme pored pozicije i normale može imati razna druga svojstva
- Pakuju se u liste, linije i lepeze da bi definisali mesh



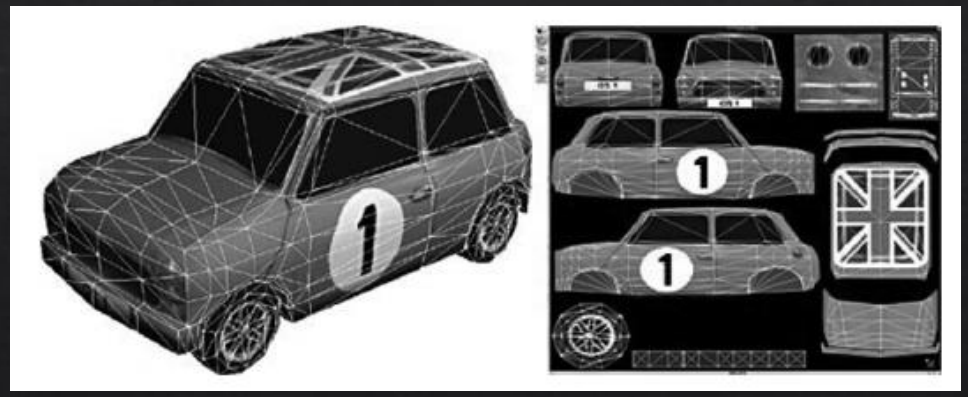




Jason Gregory. 2014. *Game Engine Architecture*

Teksture

- Koristimo ih da ne bismo za svaki vertex direktno definisali karakteristike
- Predstavljaju bitmape koje se, nalik tetovažama iz žvaka, lepe na 3D oblik – Diffuse map
- Zatim je potrebno mapirati koordinate teksture na svaki vertex



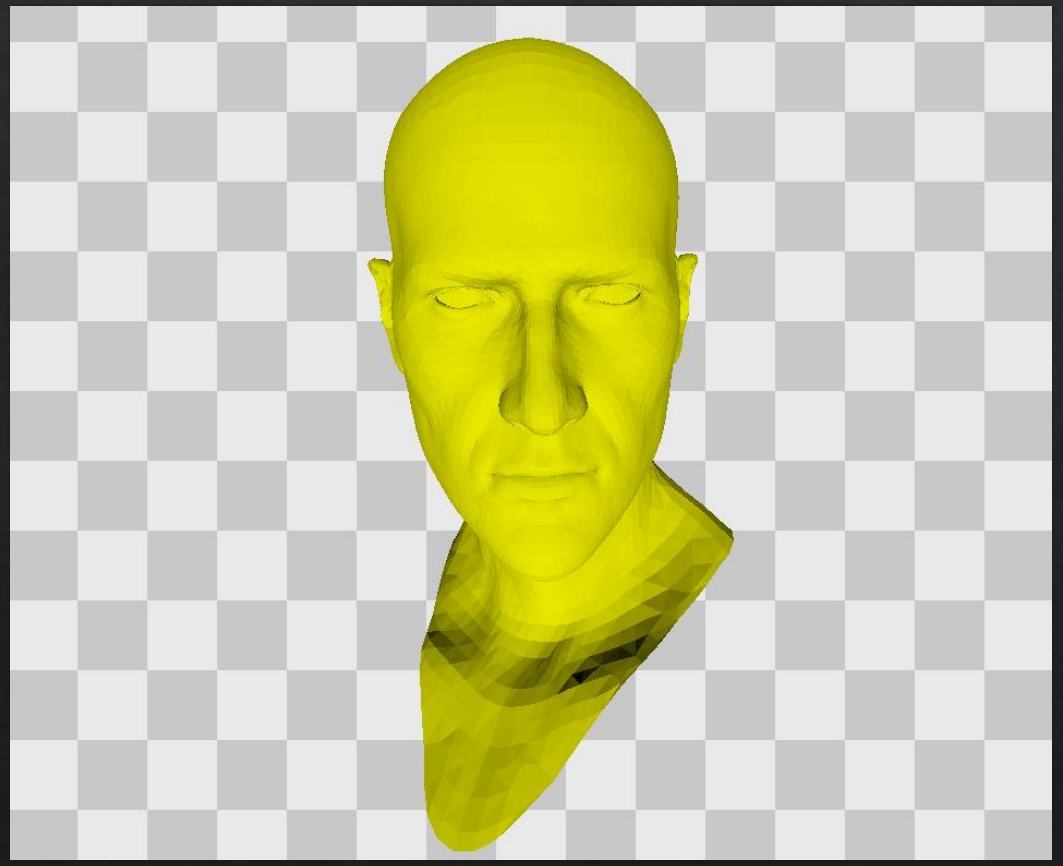
- Druge vrste tekstura: Normal maps, gloss maps, environment maps, opacity maps, LUTs

Jason Gregory. 2014. *Game Engine Architecture*

Materijal

- Kompletan opis vizuelne reprezentacije mesh-a
  - Teksture
  - Shaderi
  - Parametri shader-a

- U paru, mesh i materijal zajedno sadrže sve informacije potrebne za render objekta



https://renderdoc.org/blog/Graphics-in-Plain-Language/

https://renderdoc.org/blog/Graphics-in-Plain-Language/

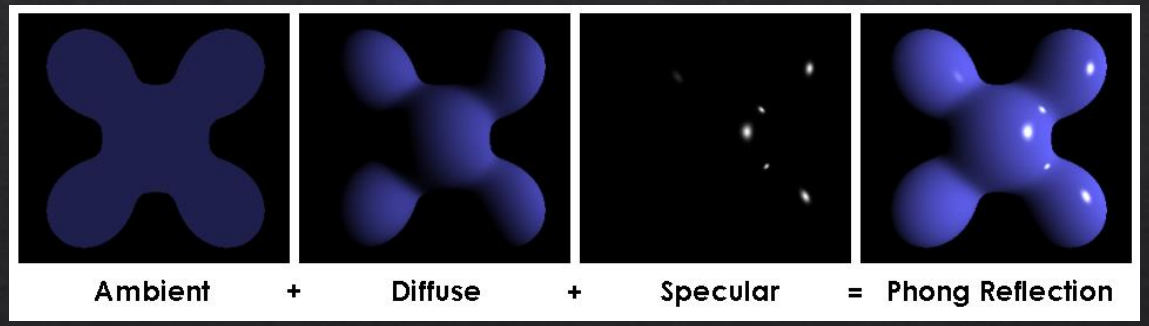
https://renderdoc.org/blog/Graphics-in-Plain-Language/

Osvetljenje

- Bez dobrog osvetljenja nema fotorealistične grafike
- Lighting je termin za procese koji izračunavaju:
  - Količinu, smer i boju osvetljenja koje pogađa određenu površinu
  - Kako se se to svetlo apsorbuje, reemituje i odbija od površine
  - Koji svetlosni zraci konačno dopiru do kamere
- Ovo izračunavanje se može vršiti u dva smera
  - Unapred, iz izvora svetla ka kameri
  - Iz kamere ka izvoru svetla
- https://learnopengl.com/Lighting/Basic-Lighting

Vrste modela osvetljenja

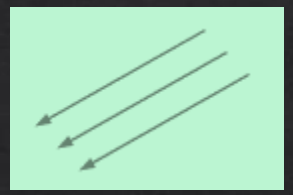
- Local illumination
  - Svetlo koje posle najviše jednog odbitka dolazi do kamere
  - Objekti na sceni ne utiču jedni na druge



- Global Illumination
  - U potpunosti matematički opisano pomoću rendering jednačine
  - Postoje razni algoritmi koji rešavaju ovu jednačinu, sa različitim pretpostavkama ili aproksimacijama
  - Značajno kompleksniji
  - U igrama se često izračunavaju offline i čuvaju kao deo geometrije
  - Ray Tracing in One Weekend - https://raytracing.github.io/

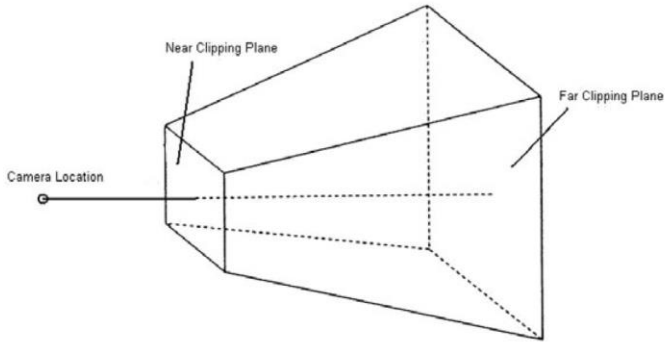
Vrste svetala

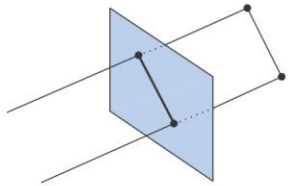
- Ambijentalno
  - Nema smer
  - Ravnomerno osvetljava celu scenu
- Usmereno
  - Beskonačno udaljeno
- Tačkasto / Point
  - Uniformno u svim smerovima počev od određene pozicije na sceni
  - Intenzitet opada sa kvadratom udaljenosti od izvora
- Spot
  - Osvetljava kupu počev od određene pozicije na sceni
  - Intenziteta opada sa kvadratom udaljenosti od izvora
  - Intenzitet opada ka unutrašnjosti kupe

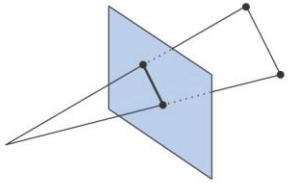


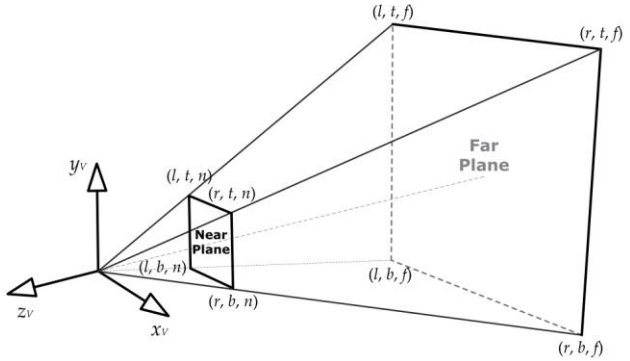
Kamera

- Nema svrhe prikazivati objekte koji se neće iscrtavati na ekranu
- View volume – prostor sa objektima koji se iscrtavaju
- Culling - proces eliminacije objekata koji su van kamere
- Kamera je opisana pozicijom u svetu, smerom gledanja, kao i matematičkim opisom view volume-a









Kamera

Jason Gregory. 2014. *Game Engine Architecture*

David H. Eberly. 2004. 3D Game Engine Architecture: Engineering Real-Time Applications with Wild Magic

Tomas Akenine-Mller, Eric Haines, and Naty Hoffman. 2018. *Real-Time Rendering, Fourth Edition* (4th. ed.)

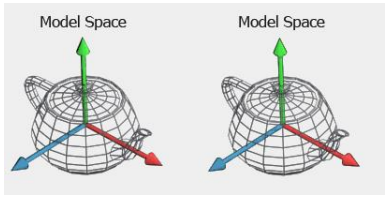
Kamera

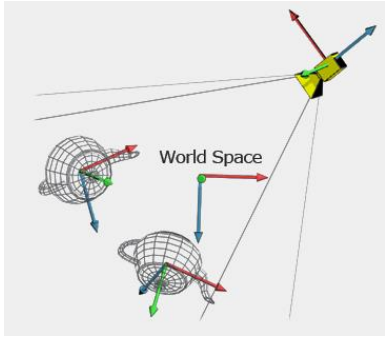
Model space - Svaki objekat, kad nastaje u nekom programu ima svoj koordinatni sistem

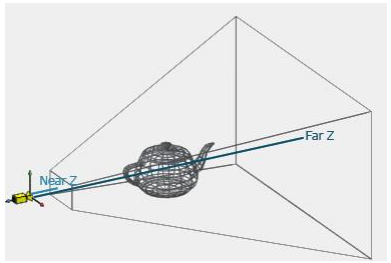
World space - Kad se objekat postavi u svet, tačke objekta se transliraju, rotiraju i skaliraju

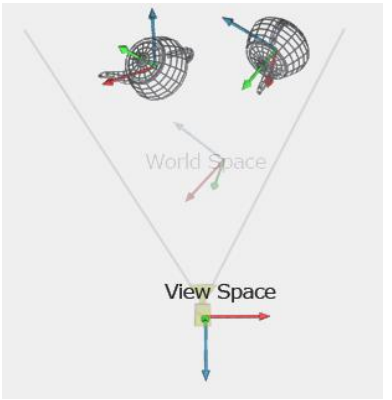
Camera space - Položaj objekata u koordinatnom sistemu kamere

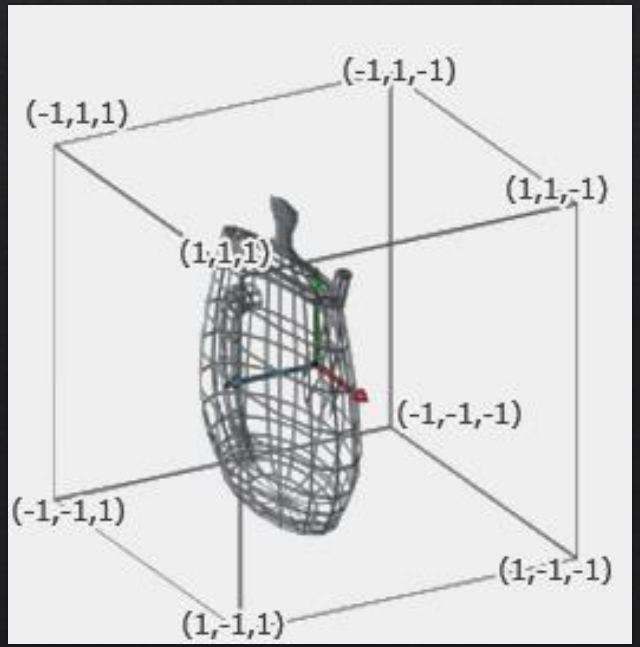
Screen space – Uzevši u obzir aspect ratio i view volume kamere, projektujemo objekte na „senzor kamere“


Model Space
Model Space


World Space


Near Z
Far Z


World Space
View Space

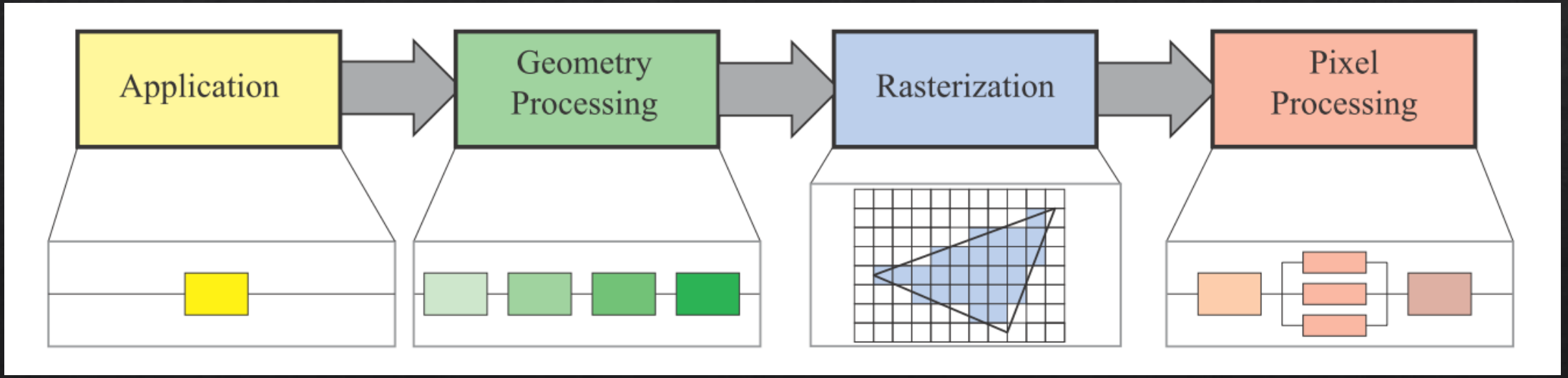

(-1,1,-1)
(-1,1,1)
(1,1,-1)
(1,1,1)
(-1,-1,-1)
(-1,-1,1)
(1,-1,-1)
(1,-1,1)

Frame buffer

- Konačna, renderovana slika se smešta u deo memorije koji se zove frame buffer
- Hardver koji prikazuje sliku čita sadržaj frame buffera 50/60/120/144 puta u sekundi
- Uglavnom postoje bar dva frame buffera koji čine Swap chain
  - Back buffer
  - Front buffer
- Postoji još bafera koji se koriste za smeštanje podataka tokom procesa renderovanja
  - Depth buffer
  - Stencil buffer

Rendering pipeline

- The main function of the pipeline is to generate, or render, a two-dimensional image, given a virtual camera, three-dimensional objects, light sources, and more.



Tomas Akenine-Mller, Eric Haines, and Naty Hoffman. 2018. *Real-Time Rendering, Fourth Edition* (4th. ed.)

Application stage

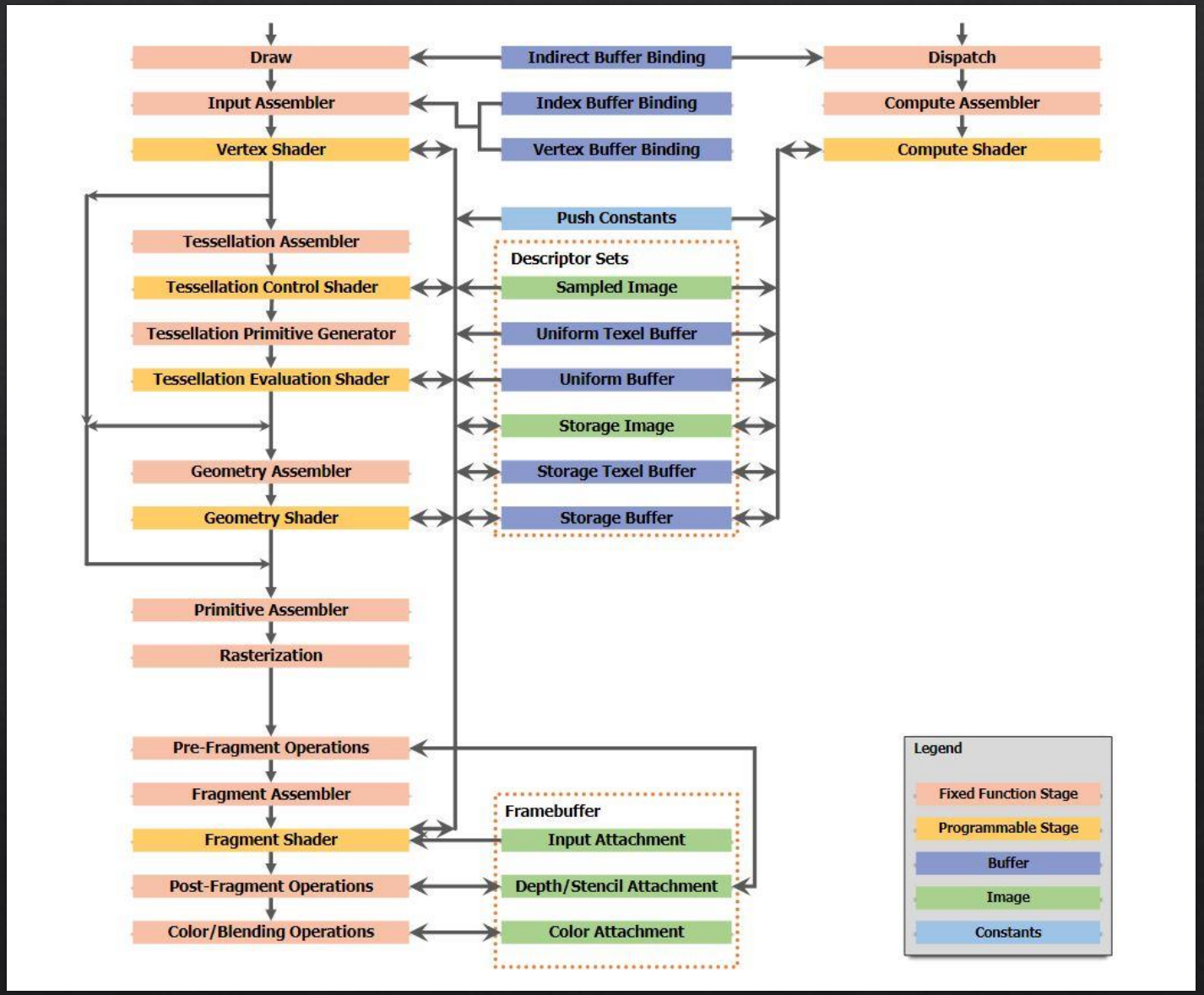
- Izvršava se na CPU
- Stvari koje utiču na rendering, ali nisu nužno deo renderinga
  - Kolizije
  - Animacije
  - Input
- Culling
  - https://docs.cryengine.com/display/SDKDOC4/Culling+Explained
  - https://www.gamesindustry.biz/articles/2016-12-07-overview-on-popular-occlusion-culling-techniques
- Slanje mesheva i materijala na GPU, konfiguracija shader-a

Culling

Culling

Geometry processing

- Vertex shader se primenjuje na svaki vertex u paraleli
  - Model -> World -> Camera -> Screen space projection
  - Opcioni post-processing
- Rasterization
  - Trougao se rastavlja na fragmente i svaki fragment predstavlja jedan piksel na ekranu
- Pixel shader se primenjuje na svaki fragment u paraleli
  - Izlaz je boja koja će se upisati u frame buffer i predstavlja boju jednog piksela



I Am Graphics And So Can You :: Part 2

Fully-procedural sea surface computing. without textures.

https://www.shadertoy.com/view/Ms2SD1

Rendering API

- Nivo apstrakcije između drajvera i korisničkog programa
  - OpenGL, DirectX 11
  - Vulkan, DirectX 12, Metal – high performance, state of the art
- Različitih nivoa kompleksnosti za programiranje
- Za multiplatformski razvoj je potrebno imati nivo apstrakcije iznad Rendering API, tako da u compile-time ili u runtime možemo da biramo koji Rendering API želimo

Rendering API

Two Sides

Renderer (API/platform agnostic)

- 2D & 3D Renderer
  - Forward, deferred, etc.
- Scene Graph
- Sorting
- Culling
- Materials
- LOD
- Animation
- Camera
- VFX
- PostFX
- Other things
  - (eg. reflections, ambient occlusion)

Render API (API/platform specific)

- Render Context
- Swap chain
- Framebuffer
- Vertex Buffer
- Index Buffer
- Texture
- Shader
- States
- Pipelines
- Render passes