Computação Gráfica para Jogos Eletrônicos

Visão geral sobre o processo de renderização de jogos digitais

Slides por: Leonardo Tórtoro Pereira (
Assistentes: Gustavo Ferreira Ceccon (gustavo.ceccon@usp.br),
Gabriel Simmel (gabriel.simmel.nascimento@usp.br) e Ítalo Tobler (italo.tobler.silva@usp.br)

Este material é uma criação do
Time de Ensino de Desenvolvimento de Jogos Eletrônicos (TEDJE)
Filiado ao grupo de cultura e extensão Fellowship of the Game (FoG), vinculado ao ICMC - USP

Objetivos

- Introduzir a área de Computação Gráfica (CG)
- Mostrar contribuições dos jogos eletrônicos para a área
- Mostrar a evolução dos *hardwares* da área
  - De CPU a GPU
  - E também do *pipeline*
- Mostrar os conceitos e algoritmos básicos por trás das principais técnicas utilizadas na área, além de exemplos de utilização
- Mostrar os estágios do *pipeline* (antigo e atual)

Índice

1. Introdução
2. CPU vs GPU
3. Tipos de imagens
4. Renderização
5. *Pipeline & Hardware*

1. Introdução

1. Introdução

- O que é Computação Gráfica (CG)?
  - Imagens e filmes criados usando computadores
  - Dados de imagem criados por computador
    - Principalmente com ajuda de softwares e hardwares gráficos especializados

1. Introdução

➔ Quais são os tópicos mais importantes da área?

- Design de interface de usuário
- Gráficos de *sprites* e de vetor
- Modelagem 3D
- *Shaders*
- Design de GPU
- Visão computacional
- Entre outros!

1. Introdução

- A CG baseia-se fortemente em 3 ramos da ciência
  - Geometria
  - Óptica
  - Física

1. Introdução

➔ É responsável por

- Exibir dados de imagem e arte efetivamente e de maneira agradável ao usuário.
- Processar dados de imagem recebidos do mundo físico

1. Introdução

- Revolucionou
  - Animação
  - Filmes
  - Publicidade
  - Design gráfico
  - Jogos Eletrônicos

1. Introdução

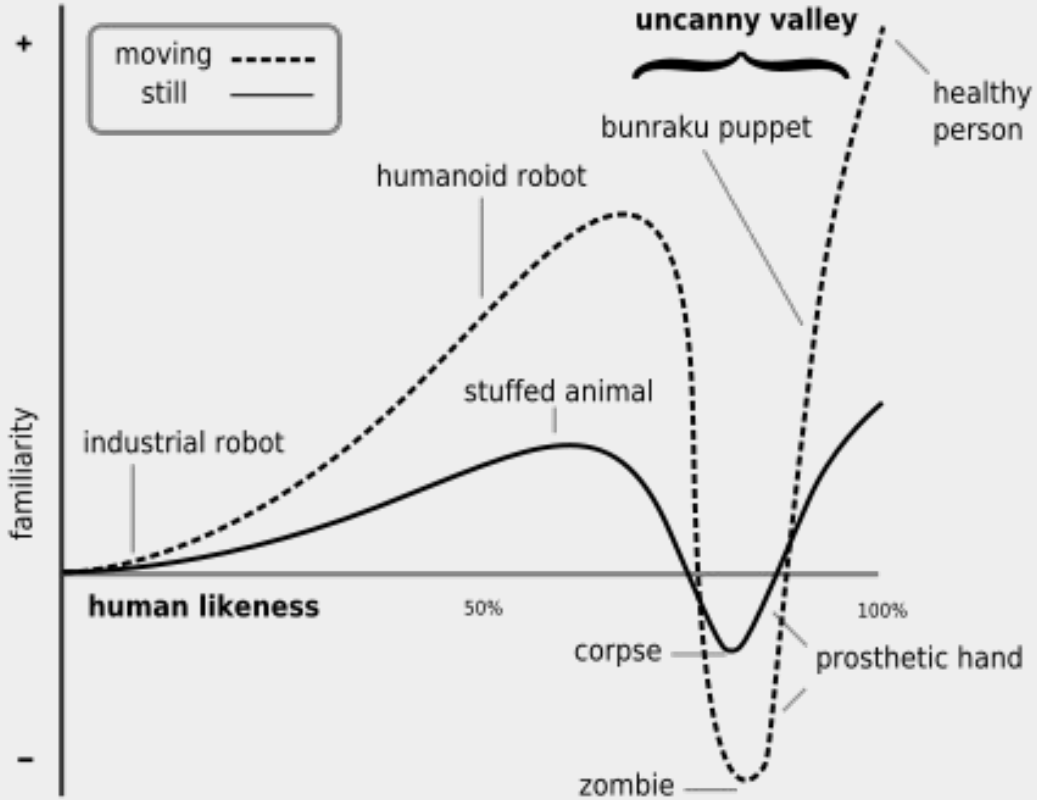
- Bibliotecas gráficas mais usadas:
  - OpenGL e DirectX
- *Trend* atual
  - Implementar OpenGL e DirectX em um *chip* especializado (Graphics Processing Unit - GPU) na placa gráfica

Bonus Stage 1: *Uncanny Valley*

Bonus Stage 1: *Uncanny Valley*

- Gráficos atuais ultra realistas
  - *Uncanny Valley*
    - Parece e move-se quase igual um ser real
      - Mas não totalmente
    - Causa asco em algumas pessoas

Uncanny Valley



https://upload.wikimedia.org/wikipedia/commons/f/f0/Mori_Uncanny_Valley.svg

Uncanny Valley

https://www.japan-zone.com/culture/bunraku.shtml

2. CPU vs GPU

2. CPU vs GPU

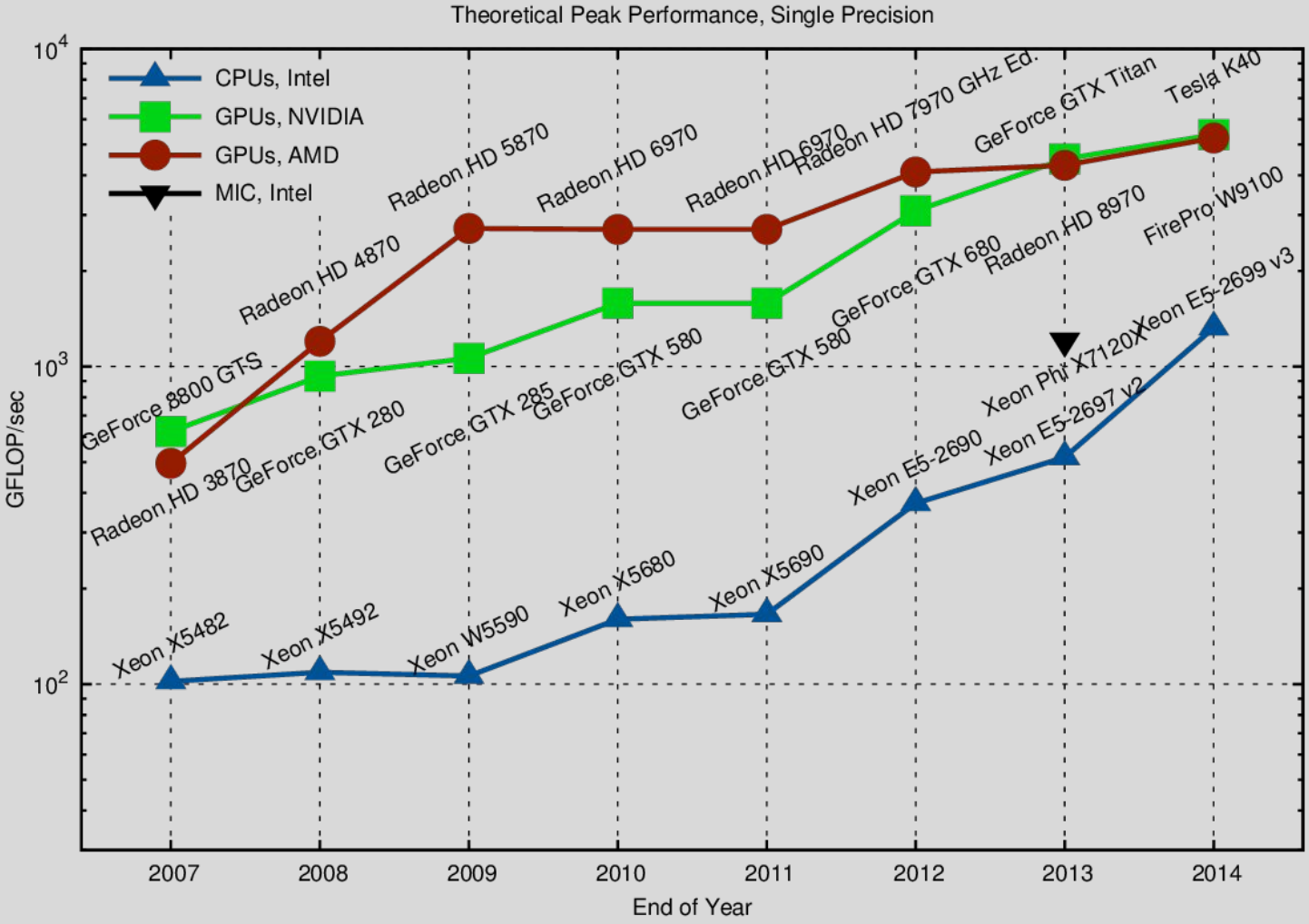
- ➔ No início da CG
  - ◆ Seu processamento era feito em CPU
    - Todo computador tem uma! :)
    - Poucos núcleos (antigamente só 1!) :(
    - Não é usada apenas para gráficos :(
    - Alto custo por núcleo :(

2. CPU vs GPU

- ➔ Atualmente
  - ◆ Seu processamento é feito (principalmente) em GPU
    - Deve ser comprada à parte :(
    - Muitos núcleos :)
      - 1152 - GTX 760
      - 2560 - GTX 1080
      - 5760 - Titan Z

2. CPU vs GPU

- ➔ Atualmente
  - ◆ GPUs são programáveis! (CUDA) :)
    - ● Novas operações adicionadas
      - ○ Podem ser usadas para aplicações não gráficas
  - ◆ FLoating-point Operations Per Second (FLOPS) alta :)



https://www.karlrupp.net/wp-content/uploads/2013/06/gflops-sp.png

2. CPU vs GPU

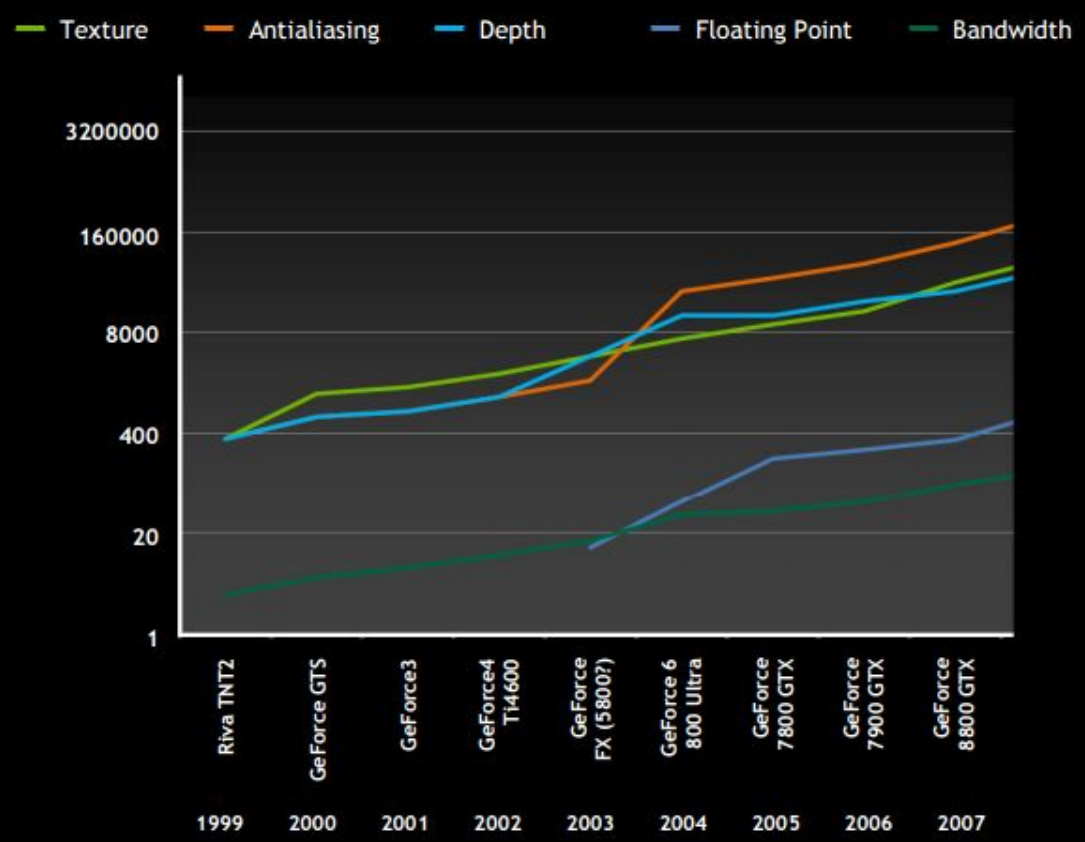
➔ Por que GPUs ficaram tão rápidas?

- Intensidade aritmética
  - Mais transistores para computações
  - Menos para lógica de decisão
- Economia
  - Demanda é alta devido à
    - Indústria multibilionária dos jogos eletrônicos
  - Mais *chips* produzidos = Menor custo

2. CPU vs GPU

- ➔ CUDA (2006)
  - ◆ Plataforma de computação paralela e também um modelo de programação
  - ◆ Criados pela NVIDIA
  - ◆ Grande aumento em performance
    - Aproveitando o poder das GPUs
  - ◆ Envia códigos C/C++, Fortran ou Python para GPU
    - Não necessita de linguagem Assembly

Performance Trends



http://www.nvidia.com/content/nvision2008/tech_presentations/Technology_Keynotes/NVISION08-Tech_Keynote-GPU.pdf

```cpp
// example1.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

#include <stdio.h>
#include <cuda.h>

// Kernel that executes on the CUDA device
__global__ void square_array(float *a, int N)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx<N) a[idx] = a[idx] * a[idx];
}

// main routine that executes on the host
int main(void)
{
  float *a_h, *a_d;  // Pointer to host & device arrays
  const int N = 10;  // Number of elements in arrays
  size_t size = N * sizeof(float);
  a_h = (float *)malloc(size);        // Allocate array on host
  cudaMalloc((void **) &a_d, size);   // Allocate array on device
  // Initialize host array and copy it to CUDA device
  for (int i=0; i<N; i++) a_h[i] = (float)i;
  cudaMemcpy(a_d, a_h, size, cudaMemcpyHostToDevice);
  // Do calculation on device:
  int block_size = 4;
  int n_blocks = N/block_size + (N%block_size == 0 ? 0:1);
  square_array <<< n_blocks, block_size >>> (a_d, N);
  // Retrieve result from device and store it in host array
  cudaMemcpy(a_h, a_d, sizeof(float)*N, cudaMemcpyDeviceToHost);
  // Print results
  for (int i=0; i<N; i++) printf("%d %f\n", i, a_h[i]);
  // Cleanup
  free(a_h); cudaFree(a_d);
}
```

- Exemplo de Cuda em C
- Função que calcula o quadrado de cada elemento de um vetor
- Programação estilo cliente/servidor
- Envia mensagem (função e parâmetros) para um núcleo da placa
- Coleta o resultado de cada núcleo
- Imprime

https://llpanorama.wordpress.com/2008/05/21/my-first-cuda-program/

2. CPU vs GPU

- ➔ CUDA pode ser usada com OpenGL
  - CUDA
    - Cálculo, geração de dados, manipulação de imagens
  - OpenGL
    - Desenha pixels ou vértices na tela
- ◆ Interoperabilidade rápida: Compartilham dados através da mesma memória no framebuffer!

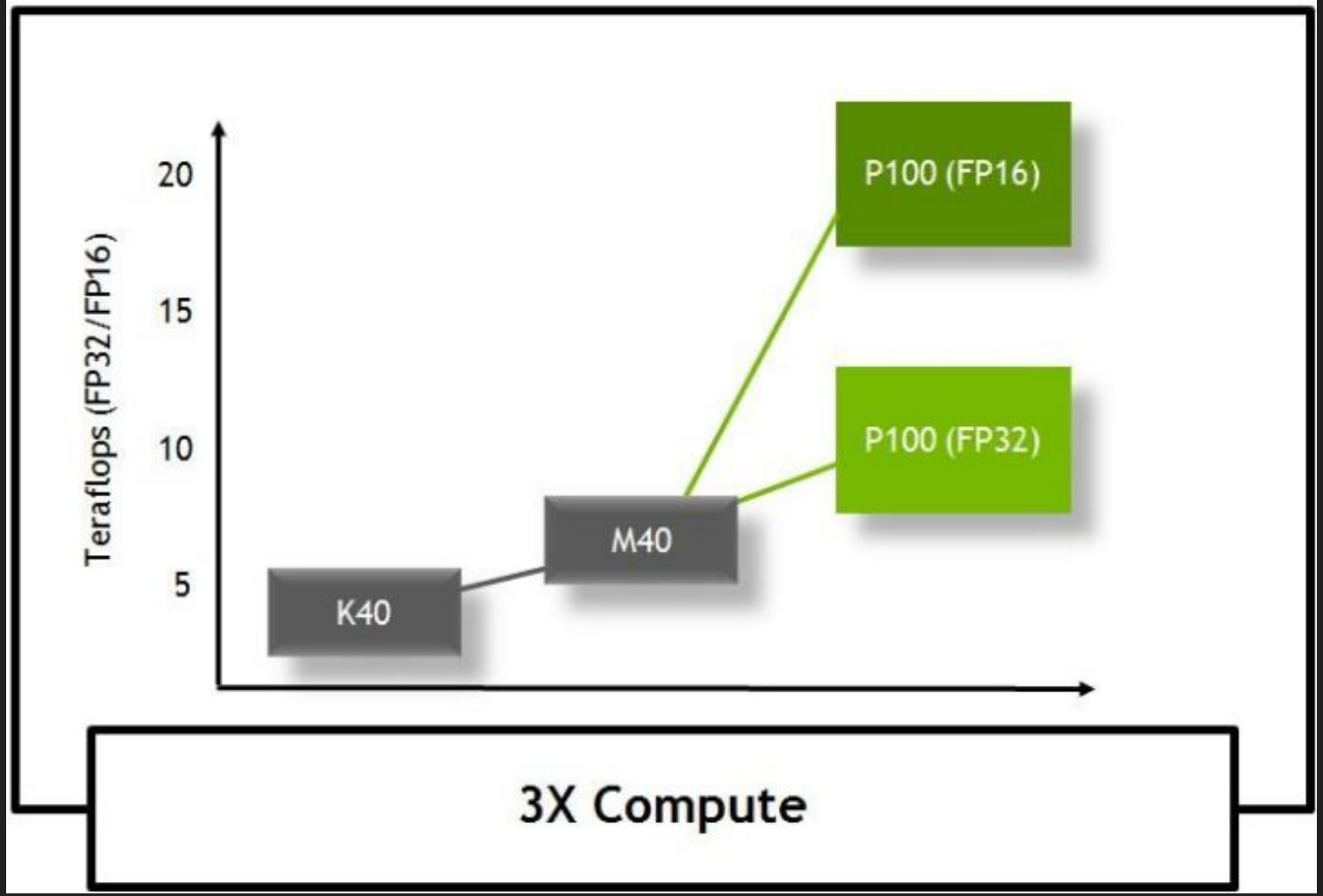
2. CPU vs GPU

- ➔ GPUs utilizadas em muitas aplicações
  - ◆ Deep Learning
  - ◆ Aplicações de Computação de alto desempenho
    - Dinâmica Computacional de Fluidos
    - Pesquisa médica
    - Visão de máquina
    - Modelagem financeira
    - Química quântica...

Bonus Stage 2: Arquitetura Pascal

Bonus Stage 2: Arquitetura Pascal

- Acelerador NVIDIA Tesla P100 utiliza a nova GPU NVIDIA Pascal™ GP100
- P100
  - 5.3 TFLOPS - pontos flutuantes de precisão dupla
  - 10.6 TFLOPS - pontos flutuantes de precisão simples
  - 21.2 TFLOPS - meia precisão
    - Pontos flutuantes de 16-bits (nativos)
    - Baixa precisão, maior poder computacional



https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

Bonus Stage 2: Arquitetura Pascal

- Deep Neural Networks (DNN)
  - GPUs aceleram aplicaçõs de 10 a 20 vezes
    - Comparadas à CPUs
    - Redução de tempo de treino de semanas para dias
  - Plataformas de computação baseadas em GPUs (NVIDIA) aceleraram, nos últimos 3 anos, o tempo de treinamento de DNNs em 50 vezes.
  - Mais de 3400 empresas usam NVIDIA para DNN

Bonus Stage 2: Arquitetura Pascal

- ➔ NVLink
  - ◆ Popularidade da computação acelerada por GPU
    - Muitos sistemas com 4-GPUs e 8-GPUs
    - 2 GPUs para cada CPU de supercomputadores
  - ◆ NVLink provê transferência de dados GPU-para-GPU com largura de banda **bidirecional** de até 160GB/s (5x a largura de banda do PCIe Gen 3x16).

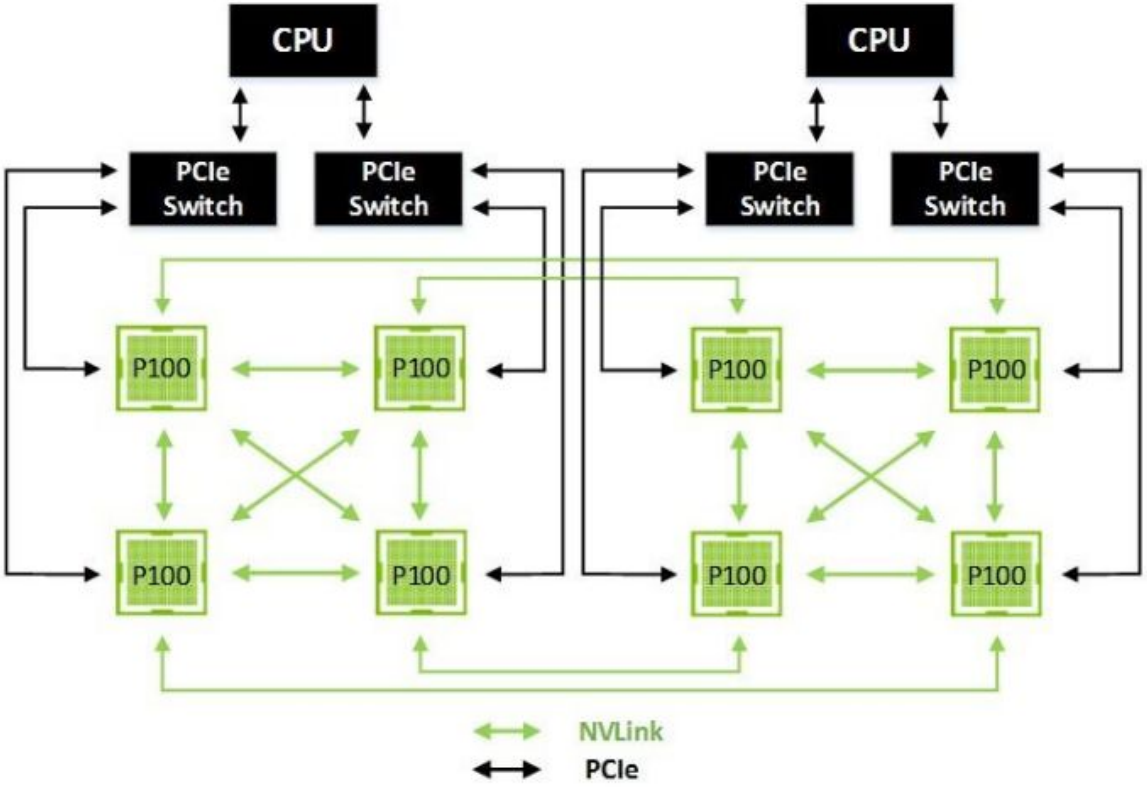


Figure 4. NVLink Connecting Eight Tesla P100 Accelerators in a Hybrid Cube Mesh Topology

https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

Bonus Stage 2: Arquitetura Pascal

- GP100
  - Memória unificada
    - CPU e GPU compartilham mesma memória
  - Foco em aplicação ao invés de alocação e transferência de dados
  - Linguagem CUDA simplificada

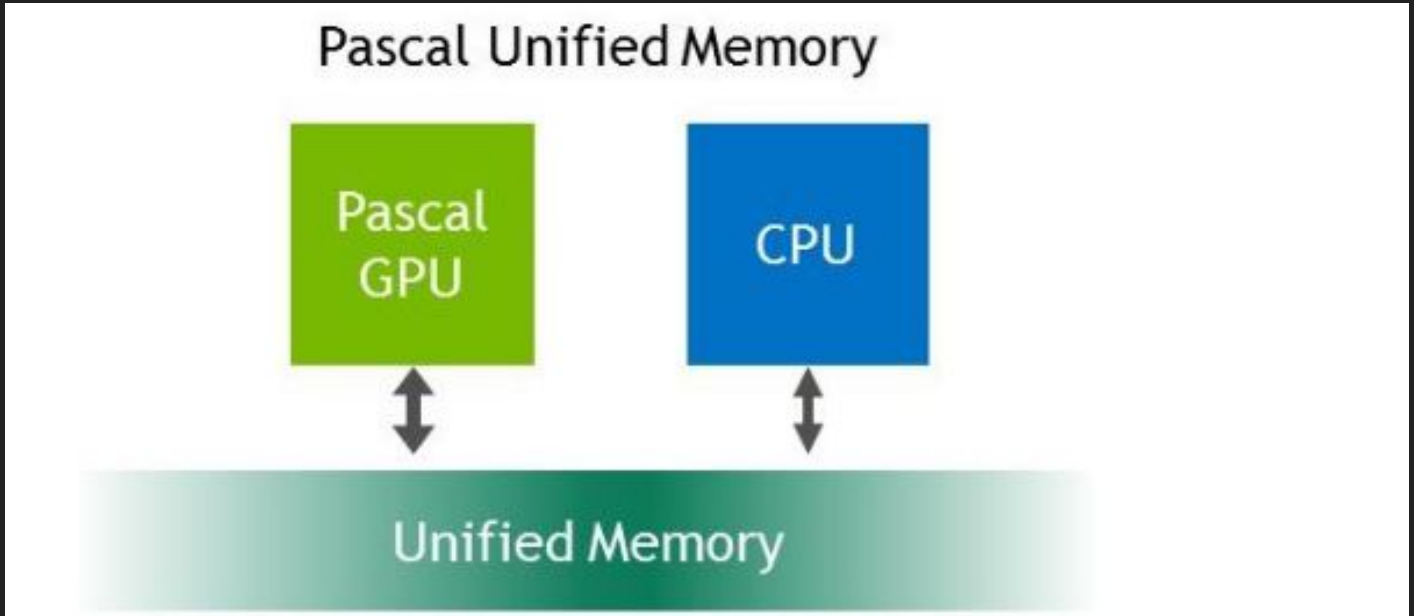


Figure 21. Pascal GP100 Unified Memory is not Limited by the Physical Size of GPU Memory.

https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

CPU Code

```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);

  qsort(data, N, 1, compare);


  use_data(data);

  free(data);
}
```

Pascal Unified Memory*

```
void sortfile(FILE *fp, int N) {
  char *data;
  data = (char *)malloc(N);

  fread(data, 1, N, fp);

  qsort<<<...>>>(data,N,1,compare);
  cudaDeviceSynchronize();

  use_data(data);

  free(data);
}
```

*with operating system support

https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf

```cpp
// example1.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"

#include <stdio.h>
#include <cuda.h>

// Kernel that executes on the CUDA device
__global__ void square_array(float *a, int N)
{
  int idx = blockIdx.x * blockDim.x + threadIdx.x;
  if (idx<N) a[idx] = a[idx] * a[idx];
}

// main routine that executes on the host
int main(void)
{
  float *a_h, *a_d;  // Pointer to host & device arrays
  const int N = 10;  // Number of elements in arrays
  size_t size = N * sizeof(float);
  a_h = (float *)malloc(size);        // Allocate array on host
  cudaMalloc((void **) &a_d, size);   // Allocate array on device
  // Initialize host array and copy it to CUDA device
  for (int i=0; i<N; i++) a_h[i] = (float)i;
  cudaMemcpy(a_d, a_h, size, cudaMemcpyHostToDevice);
  // Do calculation on device:
  int block_size = 4;
  int n_blocks = N/block_size + (N%block_size == 0 ? 0:1);
  square_array <<< n_blocks, block_size >>> (a_d, N);
  // Retrieve result from device and store it in host array
  cudaMemcpy(a_h, a_d, sizeof(float)*N, cudaMemcpyDeviceToHost);
  // Print results
  for (int i=0; i<N; i++) printf("%d %f\n", i, a_h[i]);
  // Cleanup
  free(a_h); cudaFree(a_d);
}
```

- Exemplo de Cuda em C
- Função que calcula o quadrado de cada elemento de um vetor
- Programação estilo cliente/servidor
- Envia mensagem (função e parâmetros) para um núcleo da placa
- Coleta o resultado de cada núcleo
- Imprime

https://llpanorama.wordpress.com/2008/05/21/my-first-cuda-program/

Bonus Stage 2: Arquitetura Pascal

- Vantagens da memória unificada
  - Modelo de programação e memória mais simples
  - Torna estruturas de dados complexas e classes de C++ muito mais fáceis de usar na GPU
    - Qualquer estrutura de dados hierárquica ou aninhada pode ser automaticamente acessada por qualquer processador
  - Aplicações podem operar fora do núcleo em conjuntos de dados maiores que a memória total do sistema

Bonus Stage 2: Arquitetura Pascal

- ➔ Performance através de localidade de dados
  - ◆ Ao migrar dados por demanda entre CPU e GPU, o sistema pode oferecer a performance de dados locais na GPU ao mesmo tempo que permite o acesso global a dados compartilhados
  - ◆ Funcionalidade escondida pelo driver CUDA e em tempo de execução

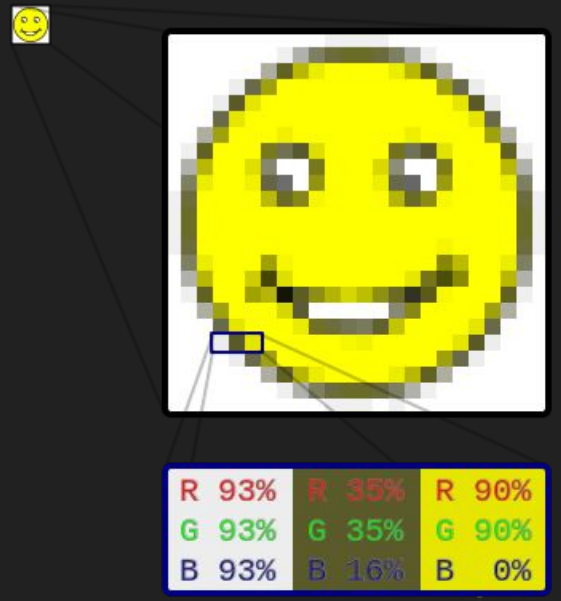
3. Tipos de Imagens

3.1. Imagens 2D

- Tipos de imagem 2D
  - Raster
  - Imagem vetorizada
  - Sprites

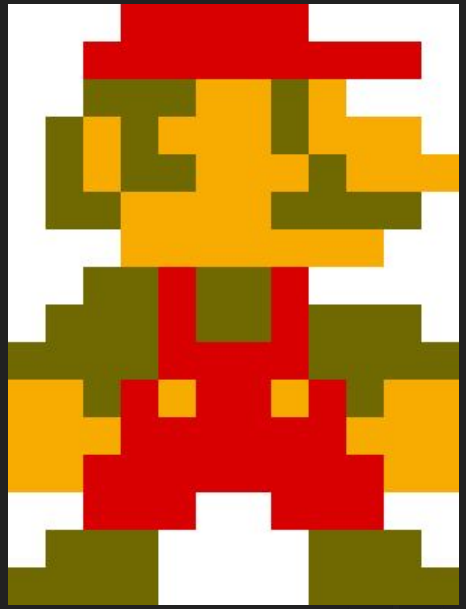
3.1.1. Raster

- Modo de representação de imagem
- Matriz de pixels
- Características do raster:
  - Altura
  - Largura
  - bits/pixel (define o alcance dos valores da matriz)



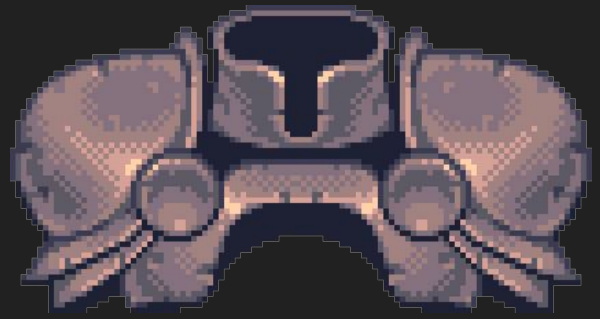
3.1.1. Raster

- Pixel Art
  - Arte a nível de pixel
  - Necessita baixa utilização de memória
    - Os primeiros consoles possuíam memória muito pequena
  - Ainda hoje utilizada em jogos
  - Baixo custo para exibição



3.1.1. Raster

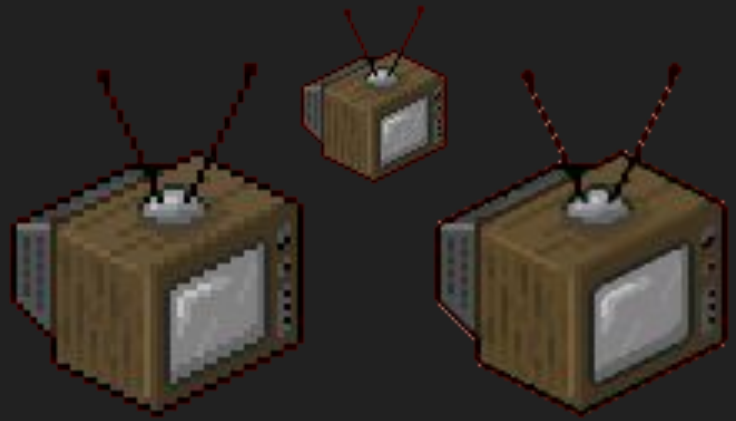
Exemplos Pixel Art

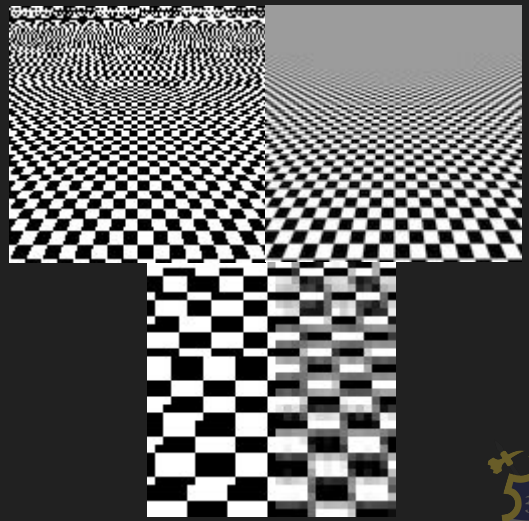


Shameless self promotion

3.1.1. Raster

- Problema: reescalar imagens
  - Algoritmos de interpolação
- Aliasing
  - Solução: Anti-aliasing



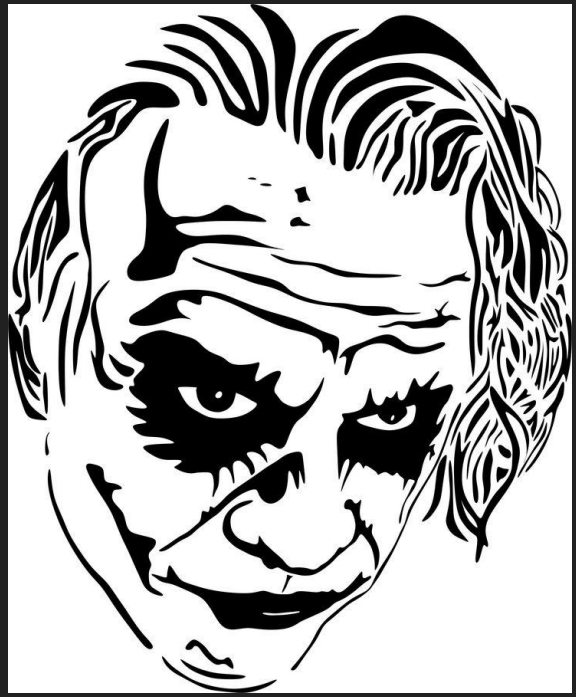


3.1.2. Imagem Vetorizada

- Representar Imagens por contornos e preenchimentos
  - Imagens compostas por “caminhos” e polígonos primitivos
- Softwares capazes de transformar imagens rasterizadas em vetorizadas
  - Grande perda de informação com imagens de tons contínuos
  - Processo inverso relativamente fácil

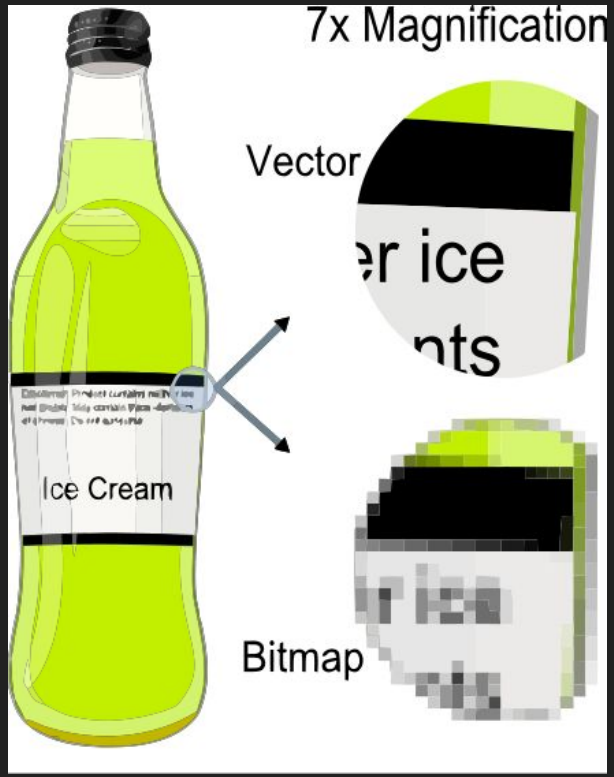
3.1.2. Imagem Vetorizada

Exemplos



3.1.2. Imagem Vetorizada

- Imagens vetorizadas não possuem problemas para reescalar
  - Pode ser rasterizada em diferentes dimensões
- Em contrapartida, a imagem precisa ser rasterizada para ser exibida e a cada vez que for reescalada



3.1.3. Sprites

- Bitmaps integrados a uma cena maior
  - Originalmente se referia a objetos independentes, processados separadamente e depois integrados a outros elementos
  - Esse método de organização facilitava detecção de colisões entre diferentes sprites

3.2. Imagem 3D

- Modelos 3D (uma malha de vértices e arestas posicionada no espaço tridimensional)
- Textura (arte plana que irá “embrulhar” o modelo, assim o colorindo)

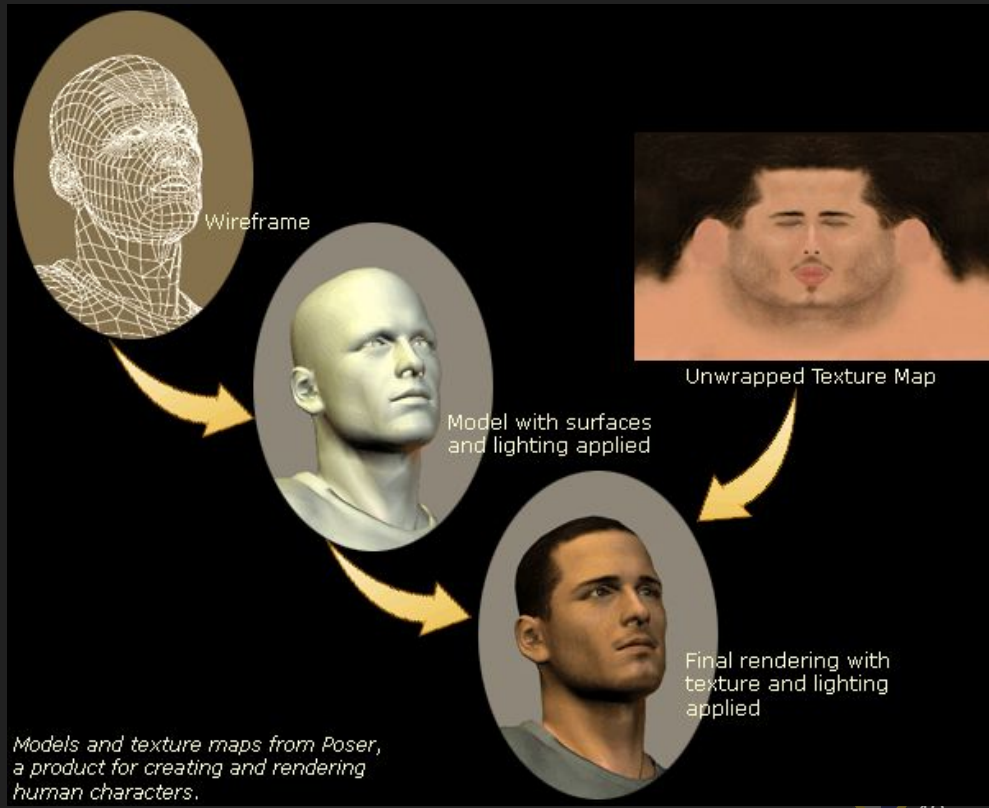
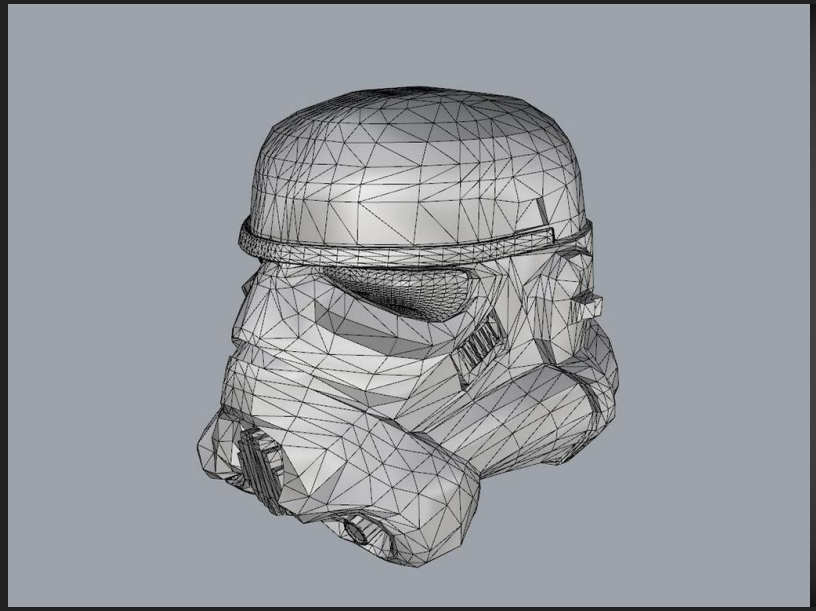


Imagem 3D Exemplos

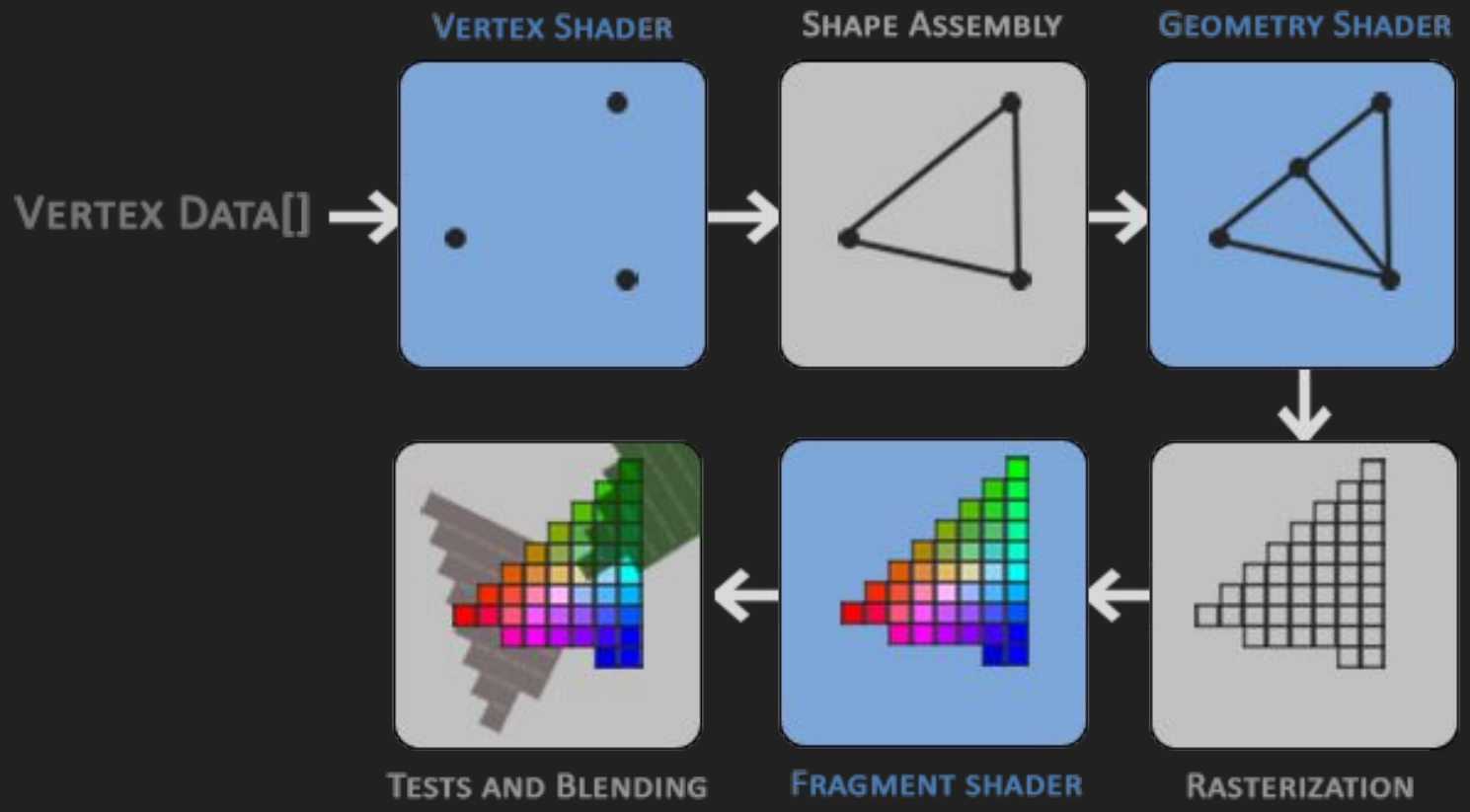


4. Renderização

4. Renderização

- Requer
  - Geometria (modelo)
  - Instruções de como desenhar (*shader*)
- Shader
  - Vertex Shader
  - Geometry Shader
  - Fragment (Pixel) Shader

Pipeline da renderização

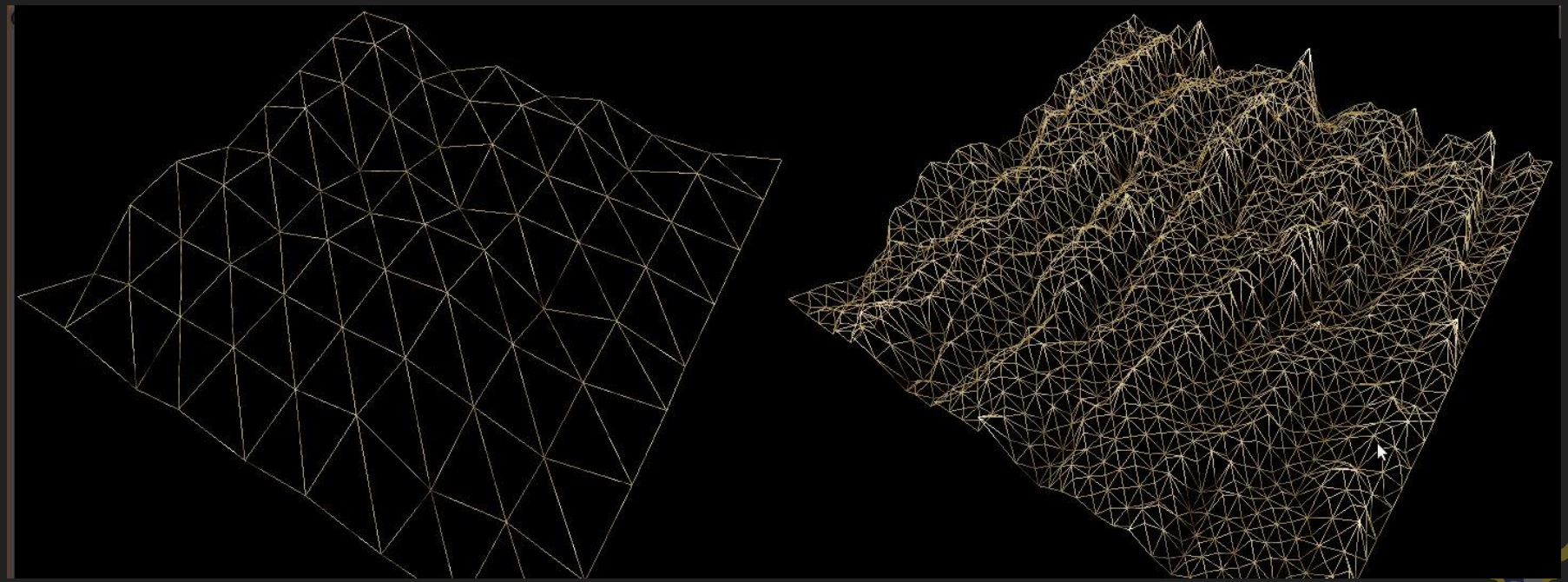


http://learnopengl.com/#!Getting-started/Hello-Triangle

4. Renderização

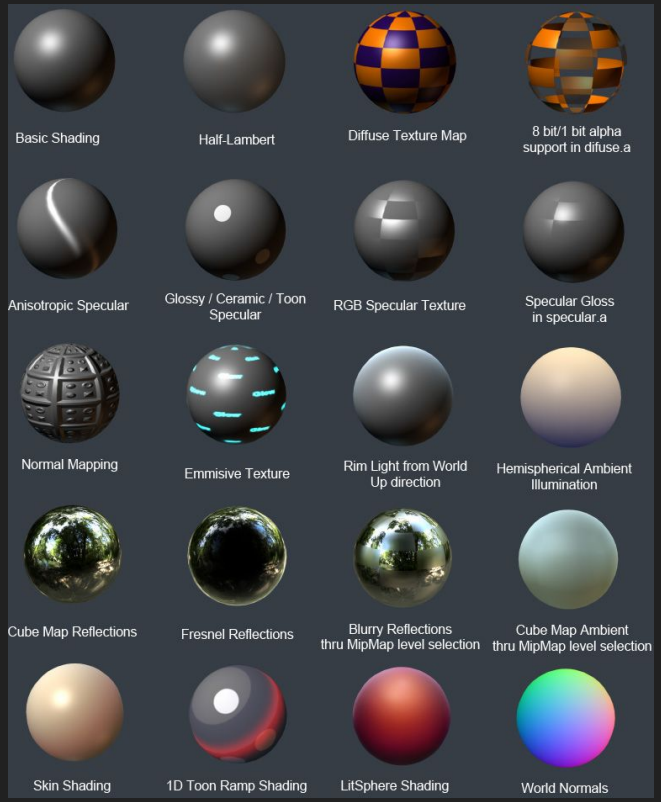
- ➔ Vertex Shader
  - ◆ Interpolação
    - Vértices
    - Cores
    - Mapas

Geometry Shader



http://irrlicht.sourceforge.net/forum/viewtopic.php?t=35500&start=15

Fragment (Pixel) Shader



http://polycount.com/discussion/70739/lcubershader-cgfx-a-maya-viewport-shader-for-fun-and-learning

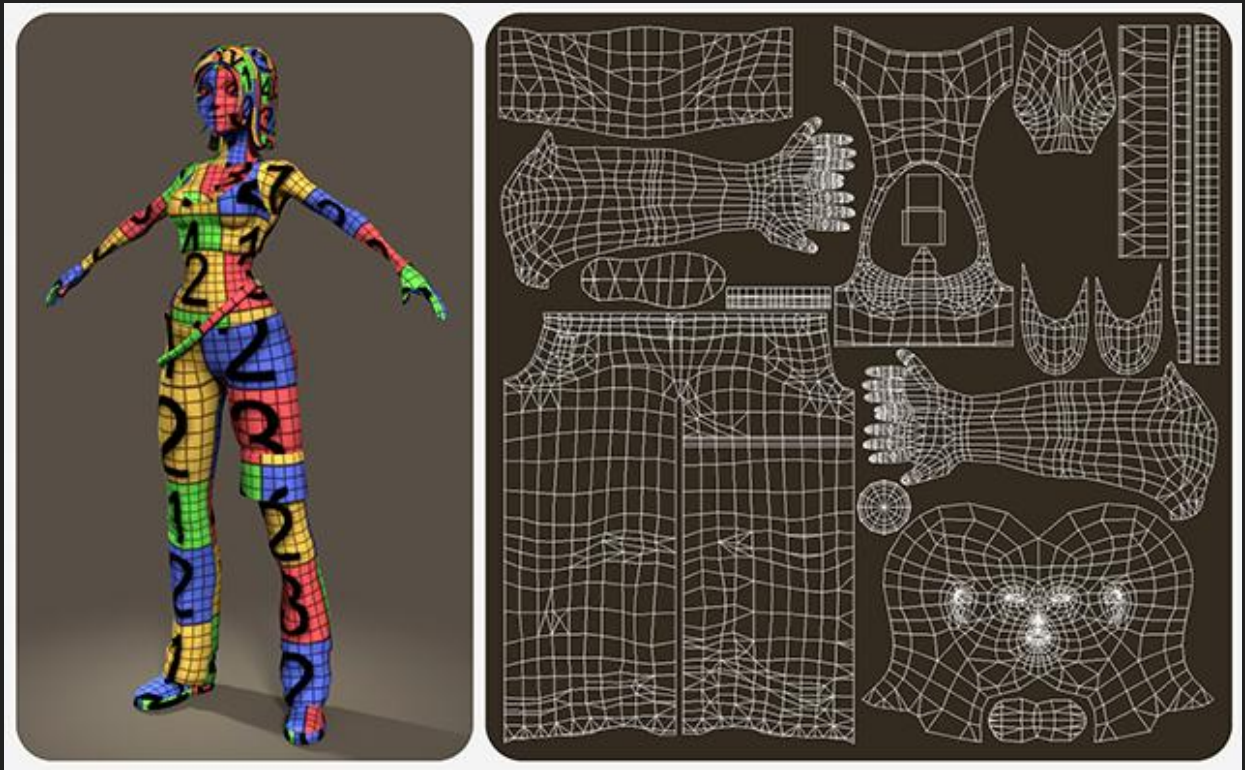
4. Renderização

- Textura
  - Mapa UV
  - Imagem
  - Filtro (interpolação)

4. Renderização

- Mapa UV
  - Mapeamento da textura de um modelo 3D para um plano 2D
  - UV = XY

Mapa UV



http://cgi.tutsplus.com/articles/game-character-creation-series-kila-chapter-3-uv-mapping--cg-26754

Imagem (Textura)

https://www.pinterest.com/pin/429953095648140137/?from_navigate=true

Filtro

- Filtros para escolher a melhor textura para dado pixel
- Dois modos comuns
  - Nearest (esquerda) seleciona o pixel que possui centro mais próximo da coordenada da textura
  - Filtro bilinear (direita) valor interpolado dos texels vizinhos da textura

http://learnopengl.com/#!Getting-started/Textures

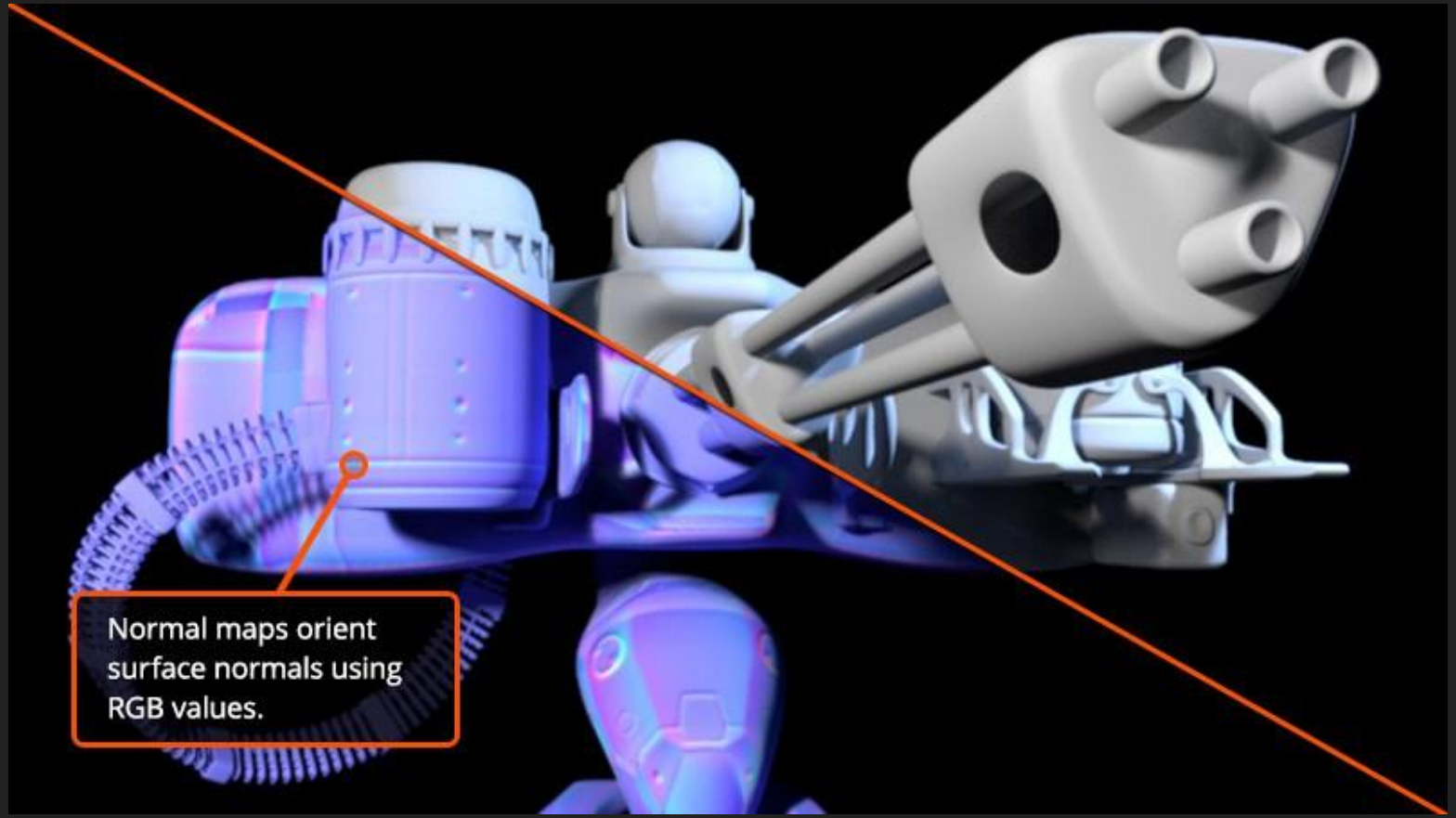
4. Renderização

- Mapeamentos (Mapping)
  - UV (já falamos)
  - Bump
    - Displacement
    - Normal
    - Parallax
    - Height
  - Cube
  - Shadow

4. Renderização

- Normal Map
  - Modifica a luz através da superfície da textura
  - Baseia-se no vetor normal à superfície

Normal Map

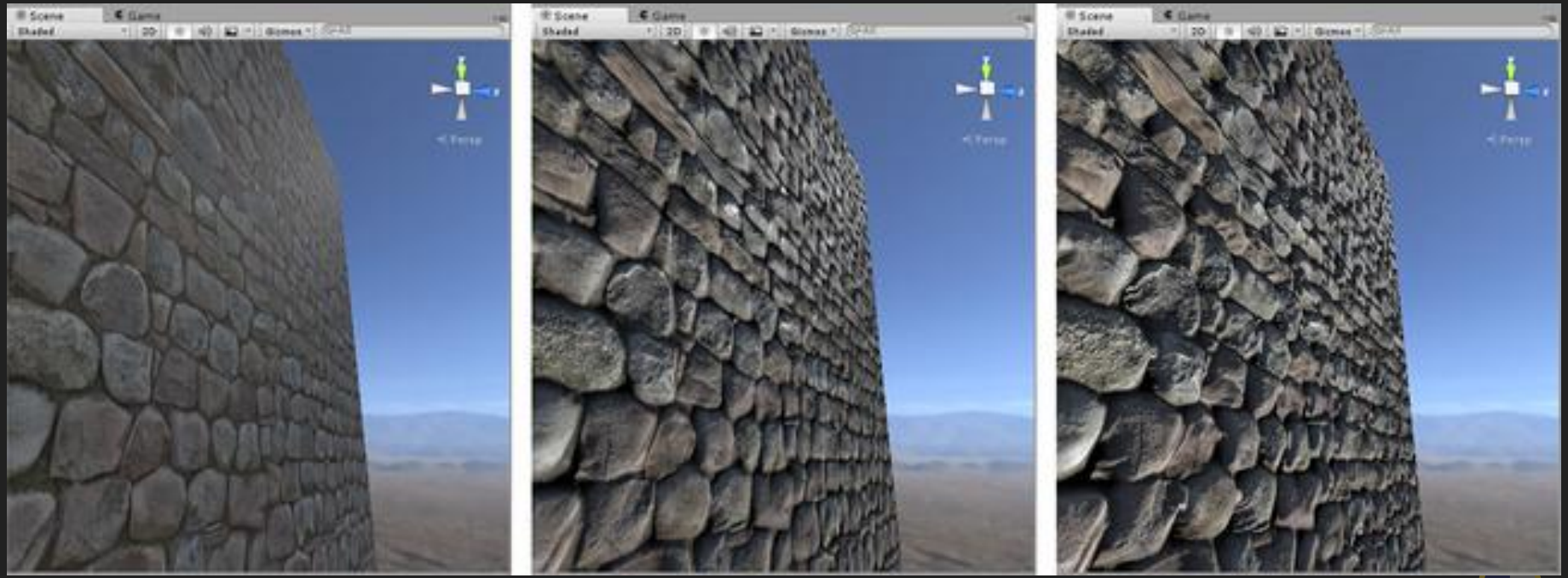


http://blog.digitaltutors.com/bump-normal-and-displacement-maps/

4. Renderização

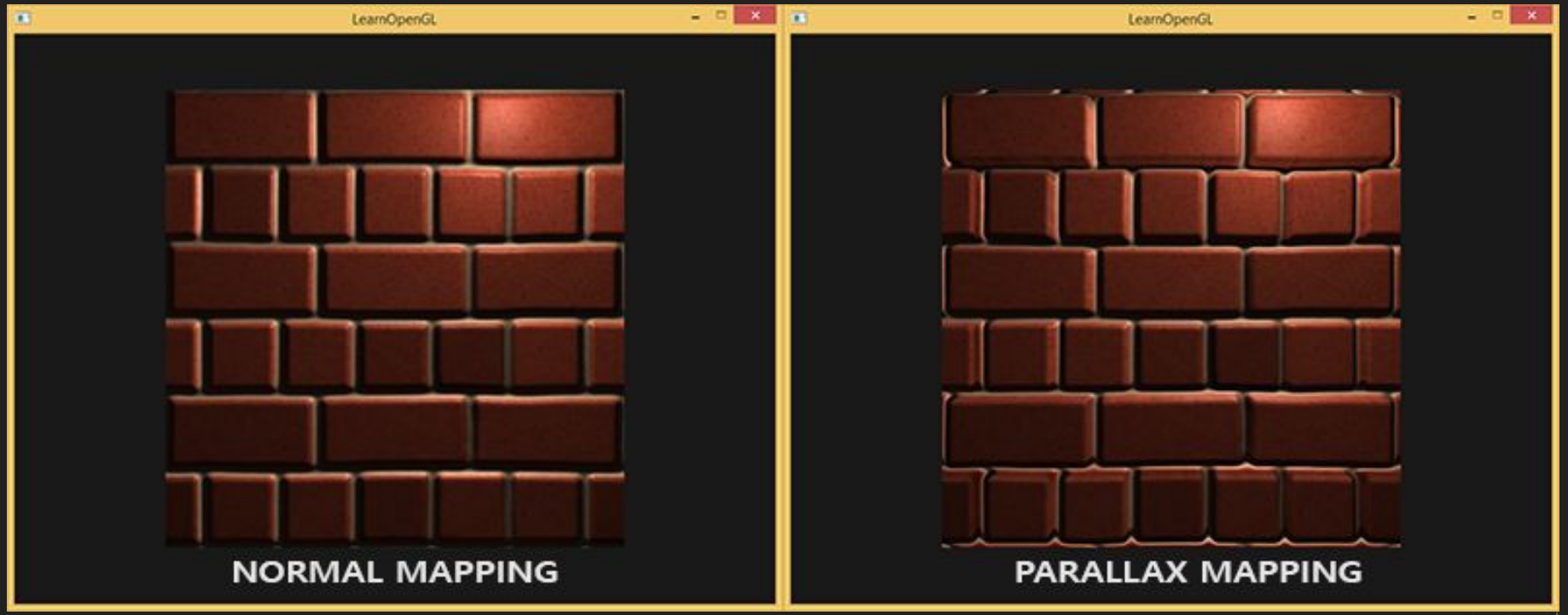
- Heigth ou Parallax map
  - Similar ao normal, mas mais complexo (e mais pesado)
  - Normalmente usado em conjunto com mapas normais
  - Move áreas da textura da superfície visível
    - Alcança um efeito a nível de superfície de oclusão
  - Protuberâncias terão suas partes próximas (frente à camera) exagerados. E a outra parte reduzida

Heightmap ou Parallaxmap



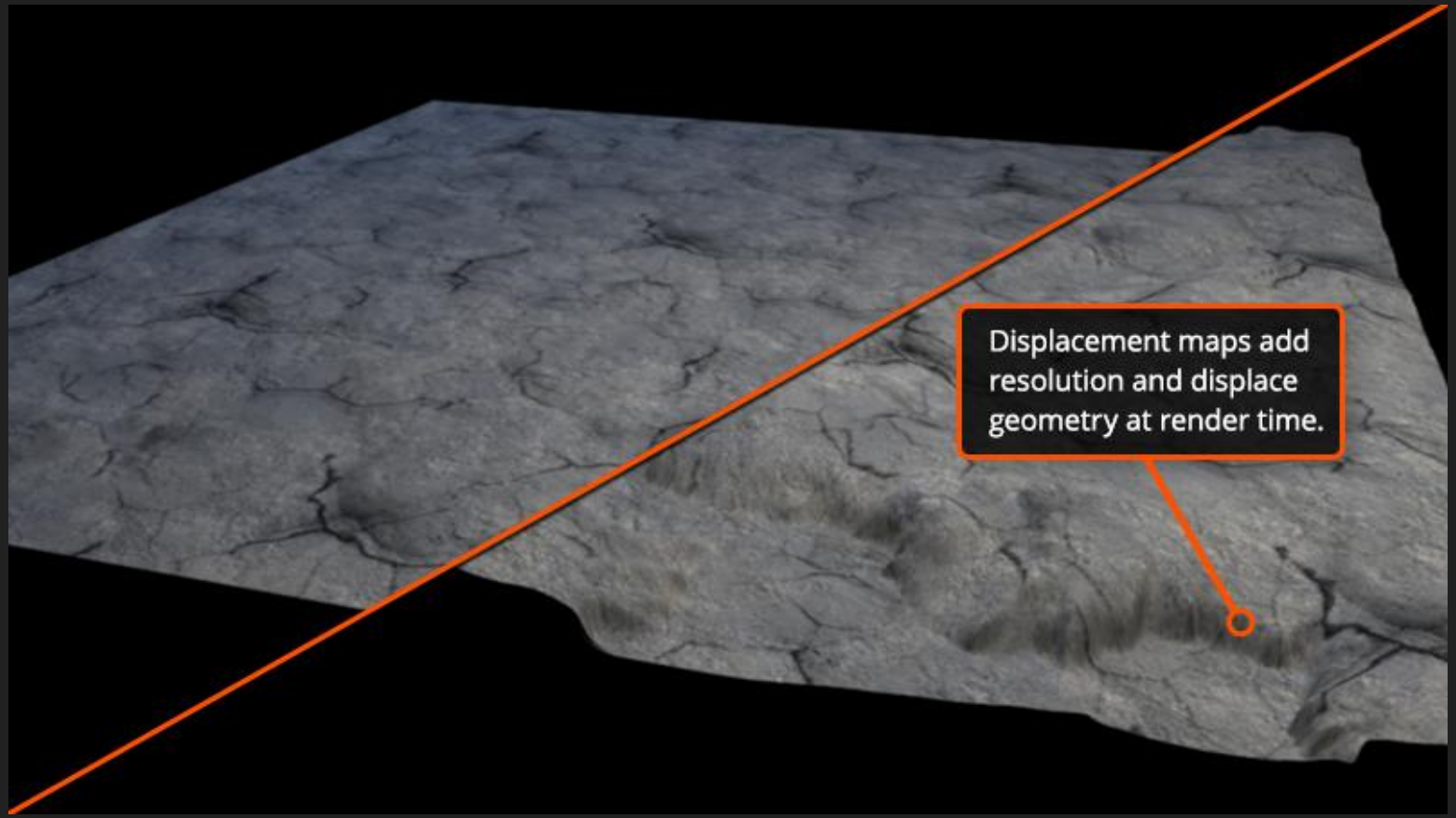
http://docs.unity3d.com/Manual/StandardShaderMaterialParameterHeightMap.html

Normal e Parallax Map



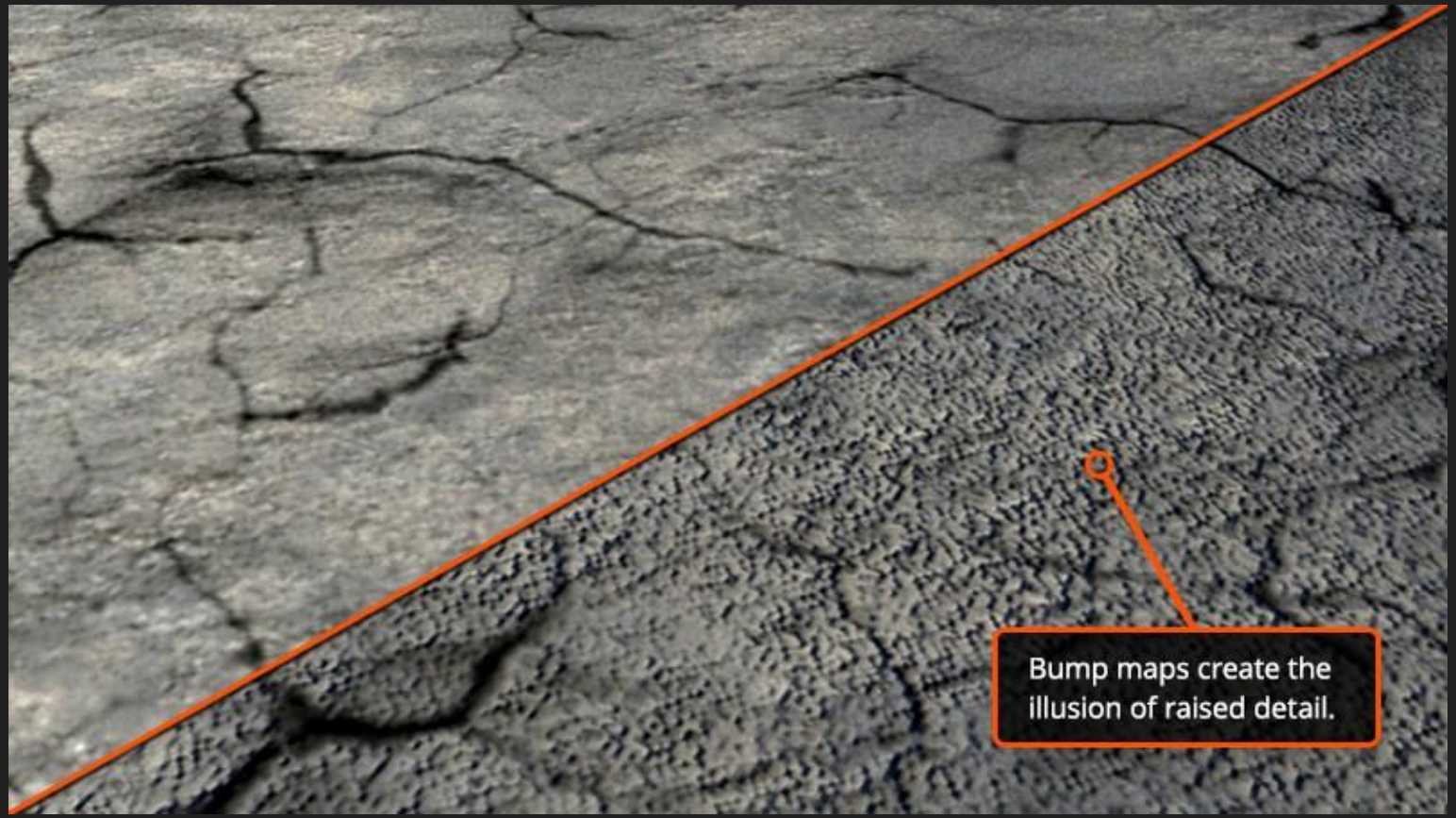
http://learnopengl.com/#!Advanced-Lighting/Parallax-Mapping

Displacement Map



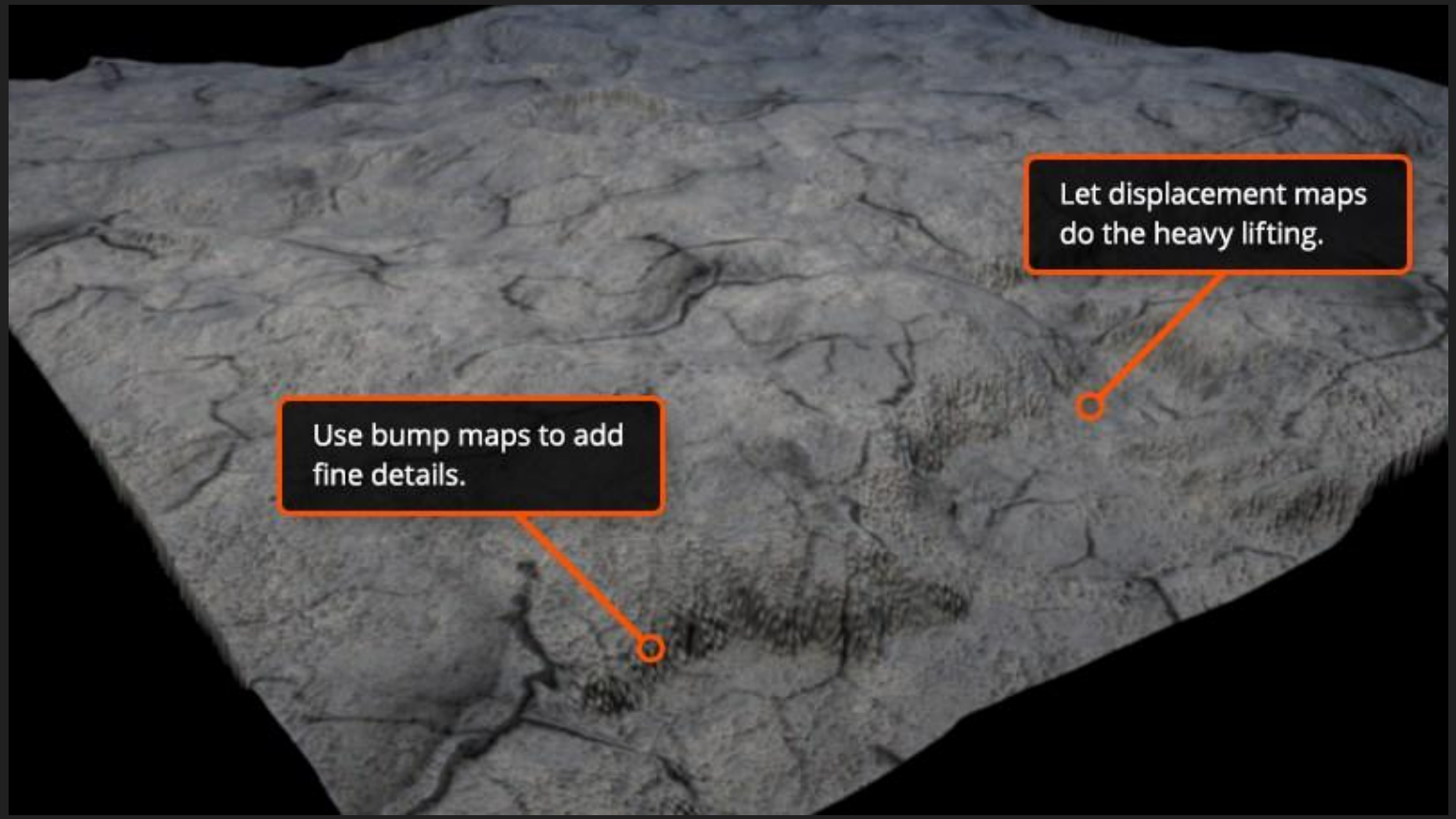
http://blog.digitaltutors.com/bump-normal-and-displacement-maps/

Bump Map



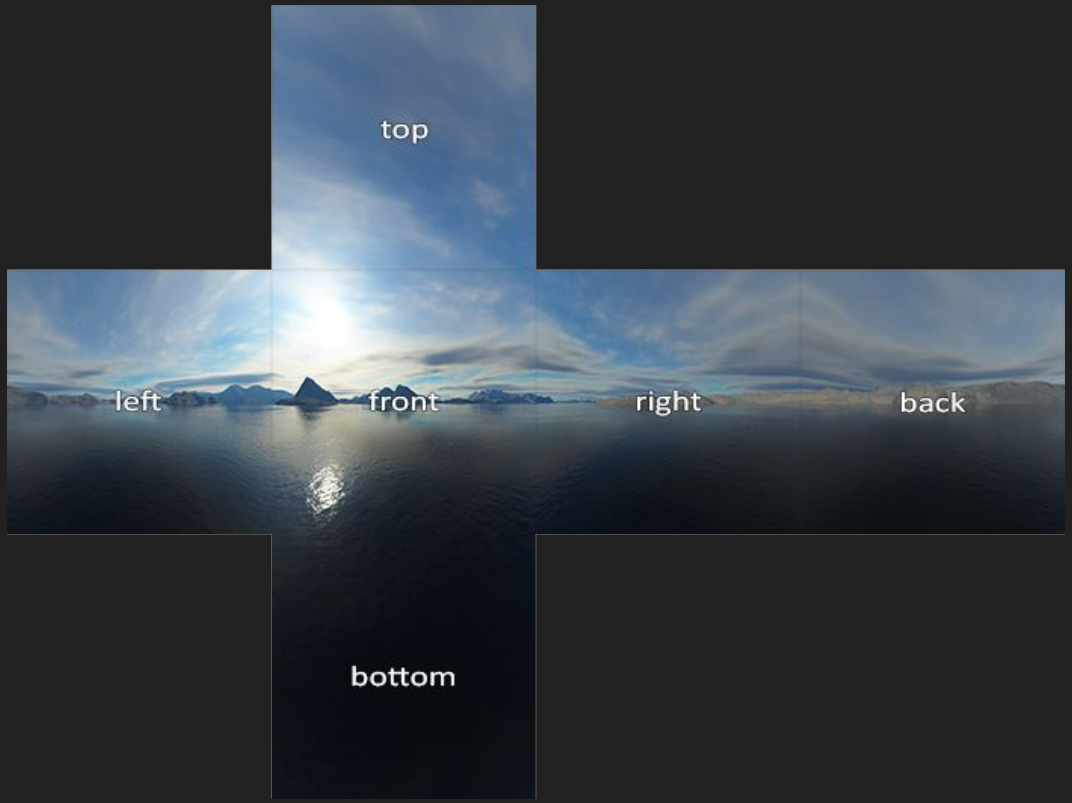
http://blog.digitaltutors.com/bump-normal-and-displacement-maps/

Bump e Displacement Maps



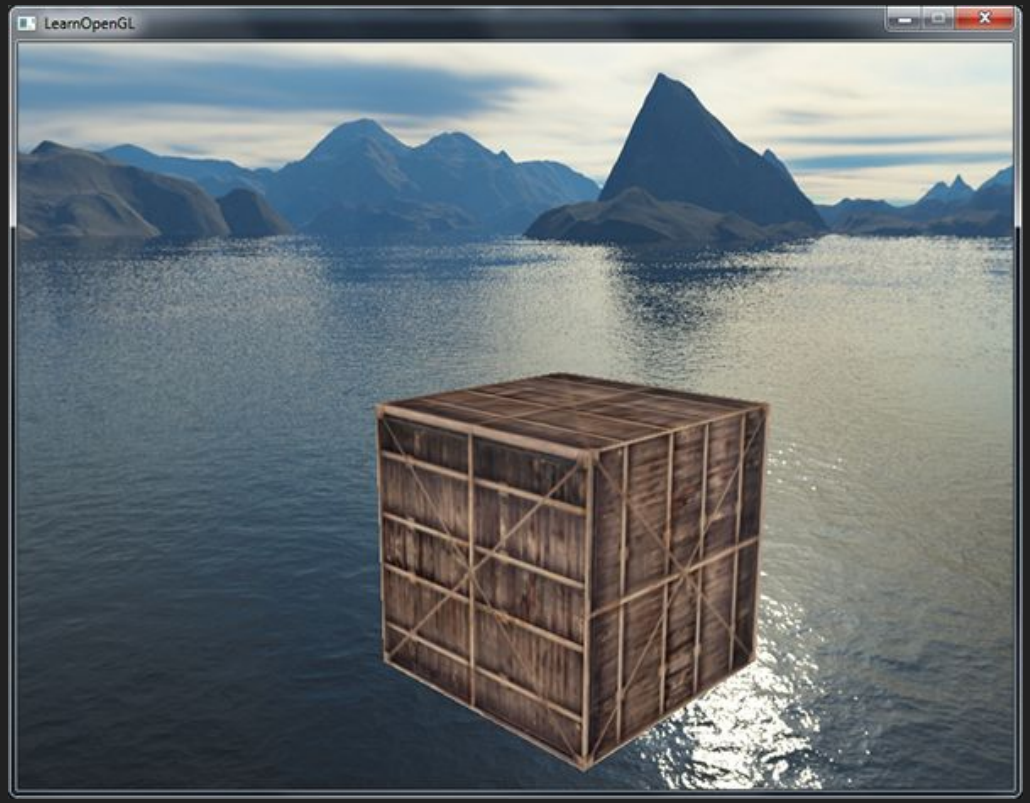
http://blog.digitaltutors.com/bump-normal-and-displacement-maps/

Cube Map



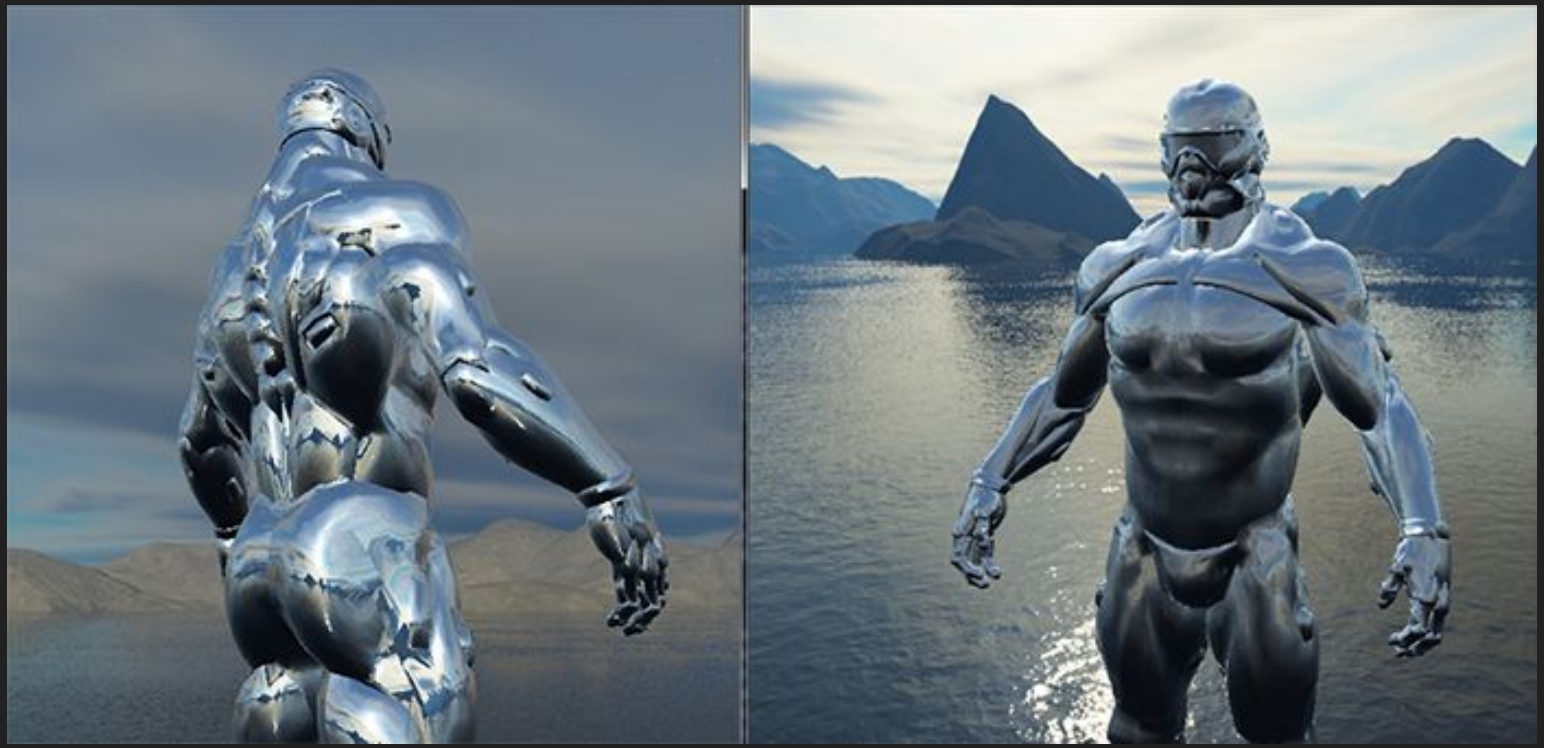
http://learnopengl.com/#!Advanced-OpenGL/Cubemaps

Skybox



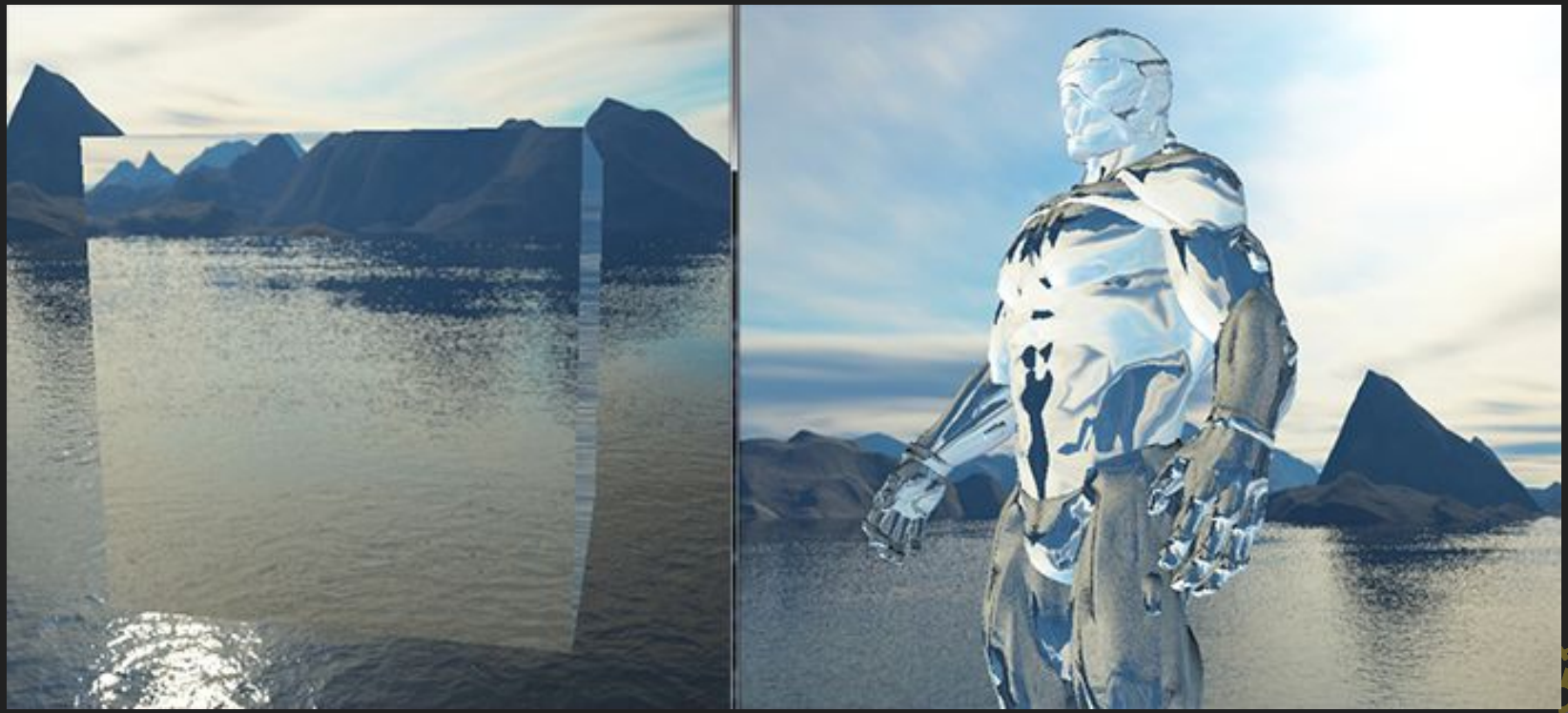
http://learnopengl.com/#!Advanced-OpenGL/Cubemaps

Reflexão



http://learnopengl.com/#!Advanced-OpenGL/Cubemaps

Refração

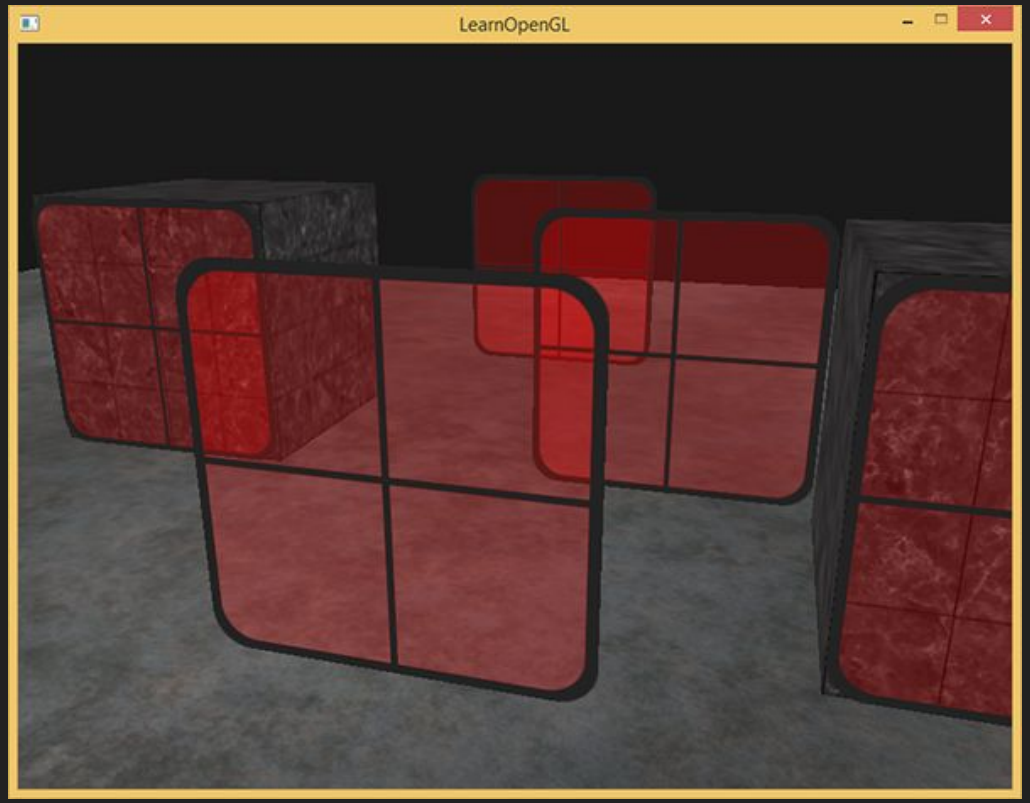


http://learnopengl.com/#!Advanced-OpenGL/Cubemaps

4. Renderização

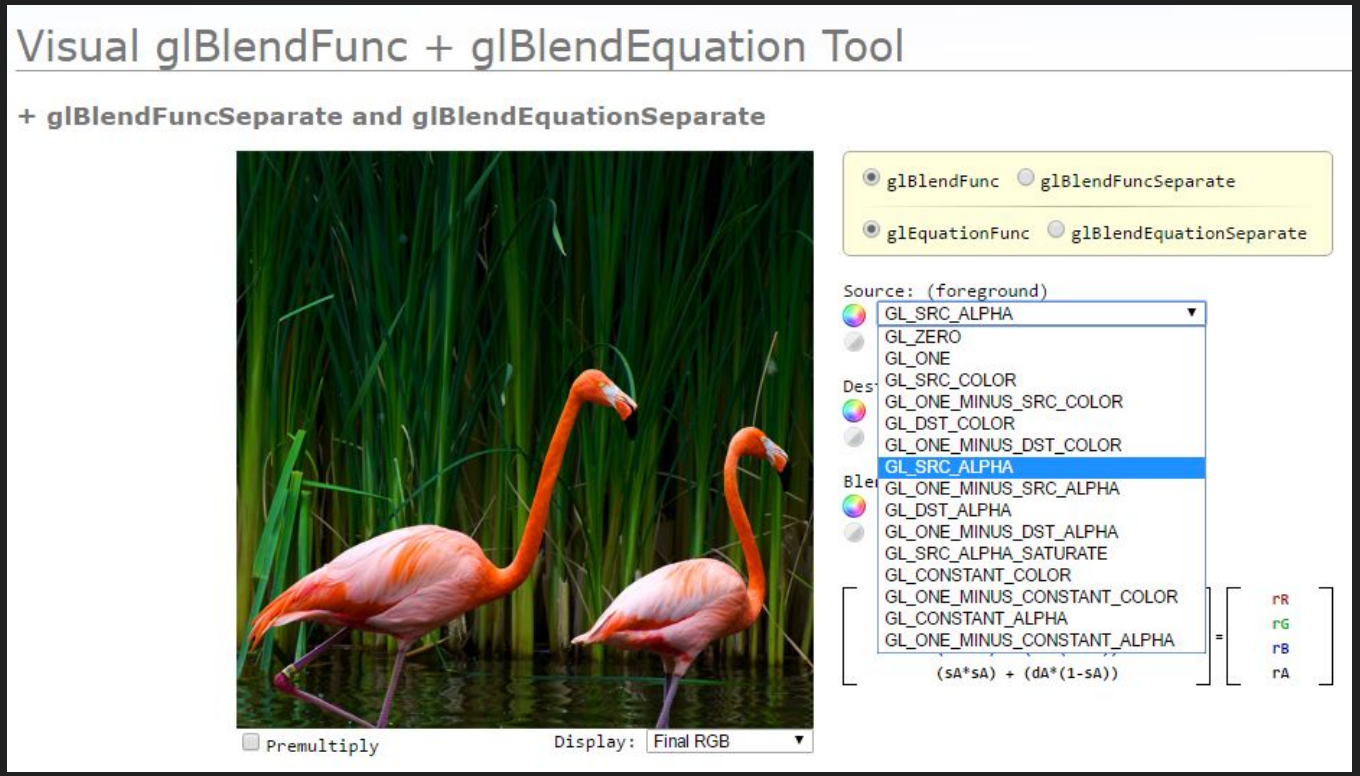
- Blending
  - Aritmética entre Alpha
  - Transparência
  - Translucência

Blending



http://learnopengl.com/#!Advanced-OpenGL/Blending

Blending

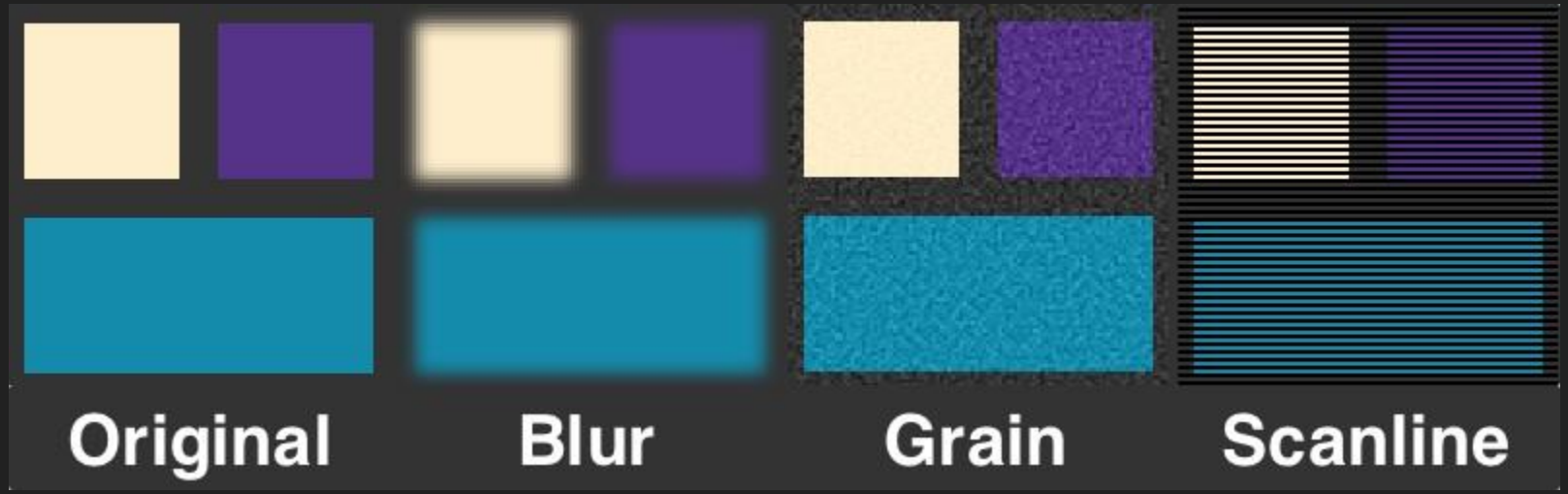


http://andersriggelsen.dk/glblendfunc.php

4. Renderização

- Post Processing
  - Fotografia
  - Aplicar *shader* no framebuffer
  - Médio custo
  - Realismo

Post Processing



http://haxepunk.com/documentation/tutorials/post-process/

Bloom

http://haxepunk.com/documentation/tutorials/post-process/

HDR

http://gamesetwatch.com/

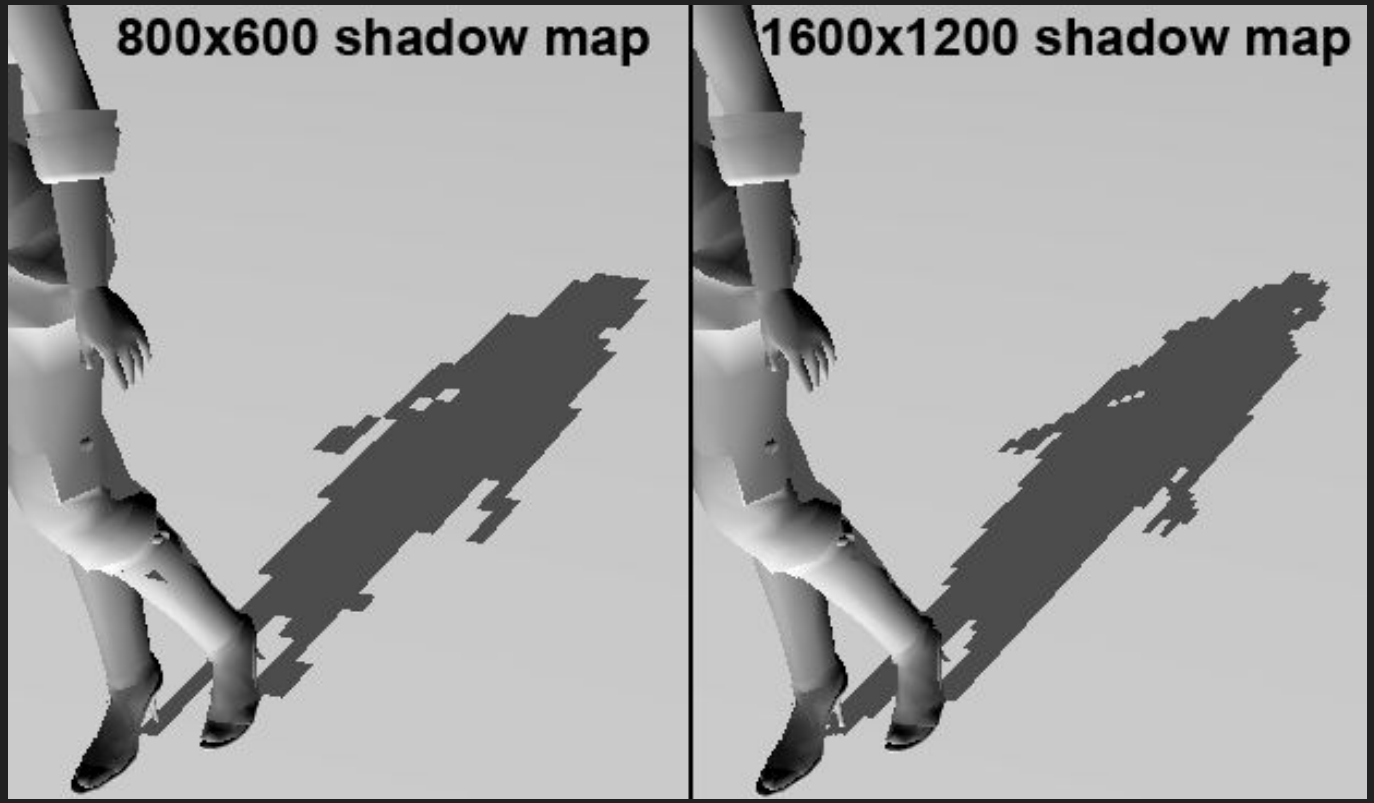
Depth of Field

https://steamcommunity.com/sharedfiles/filedetails/?id=134522361

4. Renderização

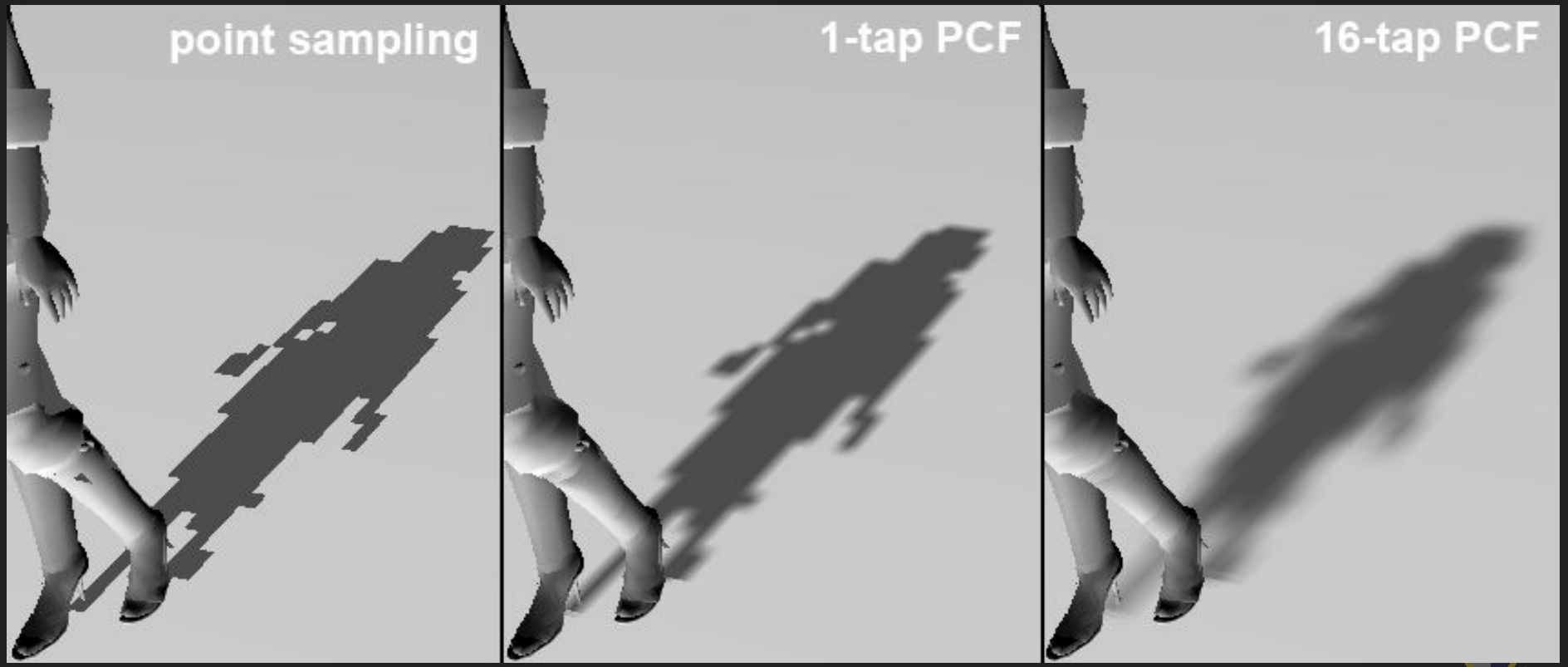
- Sombra
  - Alto custo
  - Mapeamento
  - Aproximação
  - Anti-aliasing

Shadow Map



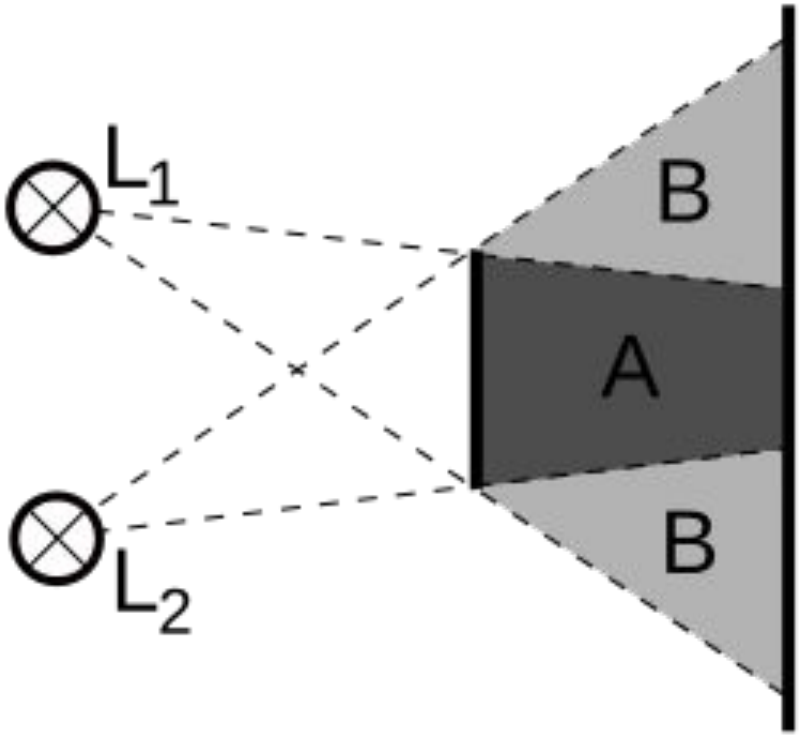
https://takinginitiative.wordpress.com/2011/05/25/directx10-tutorial-10-shadow-mapping-part-2/

Shadow Map



https://takinginitiative.wordpress.com/2011/05/25/directx10-tutorial-10-shadow-mapping-part-2/

Soft Shadow e Penumbra



https://en.wikipedia.org/wiki/Umbra,_penumbra_and_antumbra

Iluminação especular vs difusa

- Especular
  - Cor de destaque de um objeto.
  - Aparece como reflexão da luz na superfície
- Difusa
  - Cor que objeto recebe sob luz direta
  - Mais forte na direção da luz e esmaece conforme o ângulo da superfície aumenta
- https://clara.io/learn/user-guide/lighting_shading/materials/material_types/webgl_materials

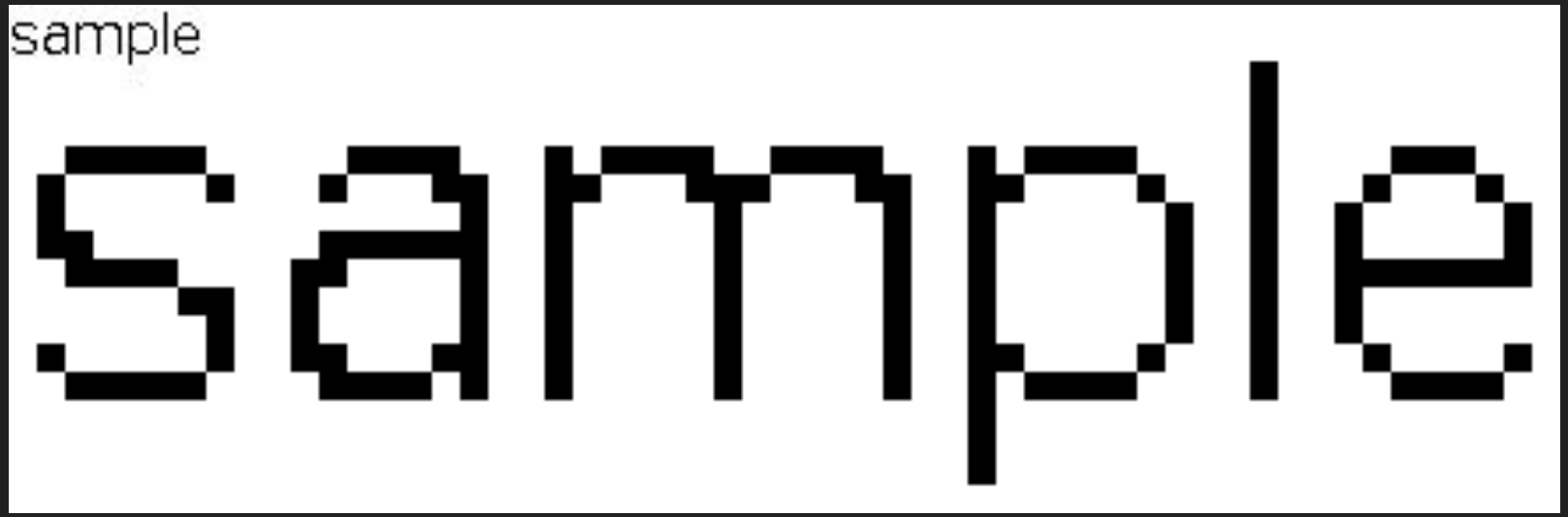
4. Renderização

- Renderizando
  - Rasterização
  - Ray casting
    - Um raio por pixel
  - Ray tracing
    - Ray casting avançado
      - Física (reflexão, refração, etc.)

4. Renderização

- http://acko.net/files/fullfrontal/fullfrontal/webglmath/online.html

Rasterização simples



https://en.wikipedia.org/wiki/Font_rasterization

Rasterização com anti-aliasing

https://en.wikipedia.org/wiki/Font_rasterization

Ray tracing

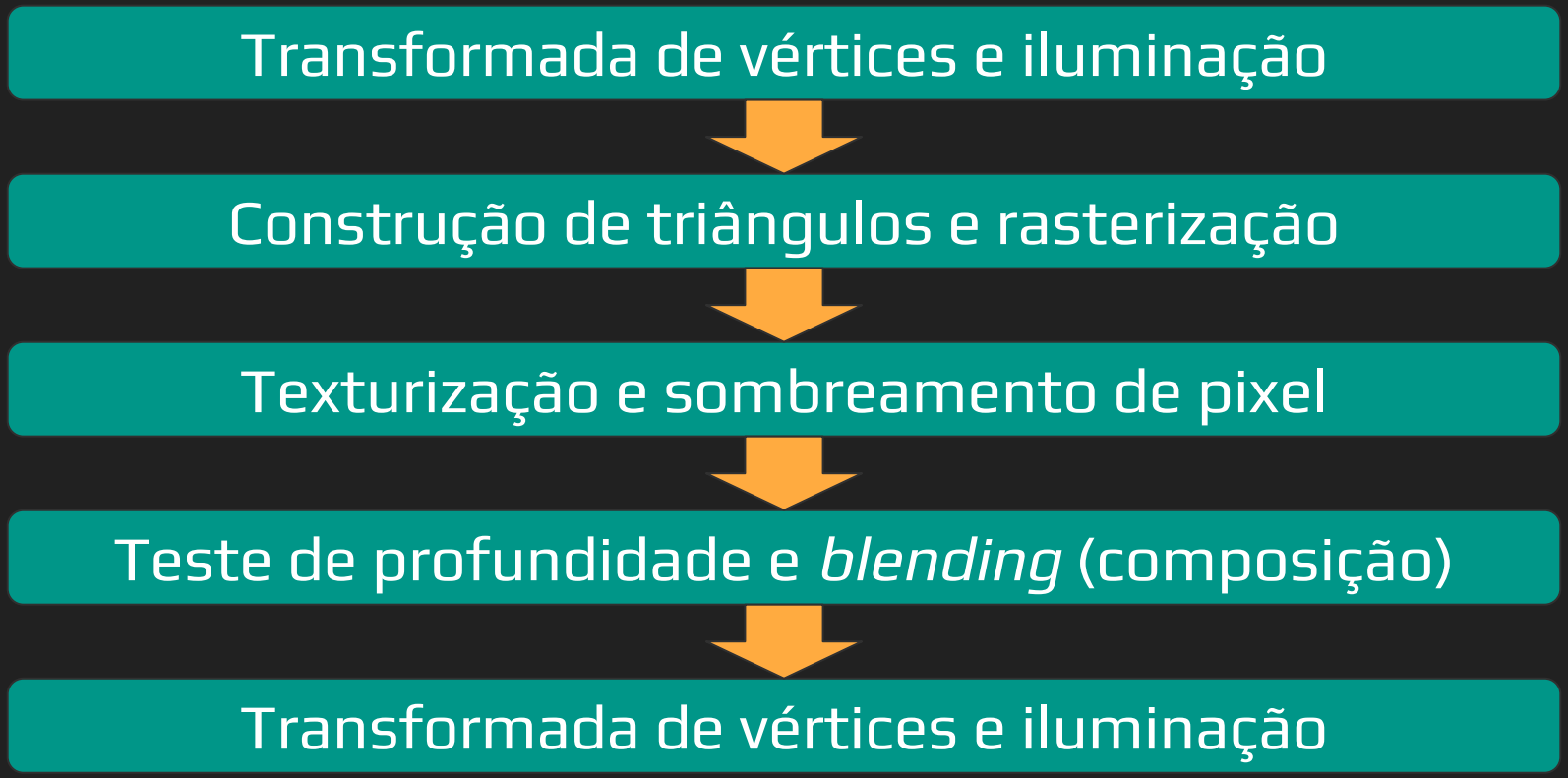
https://en.wikipedia.org/wiki/Ray_tracing_(graphics)

5. *Pipeline & Hardware*

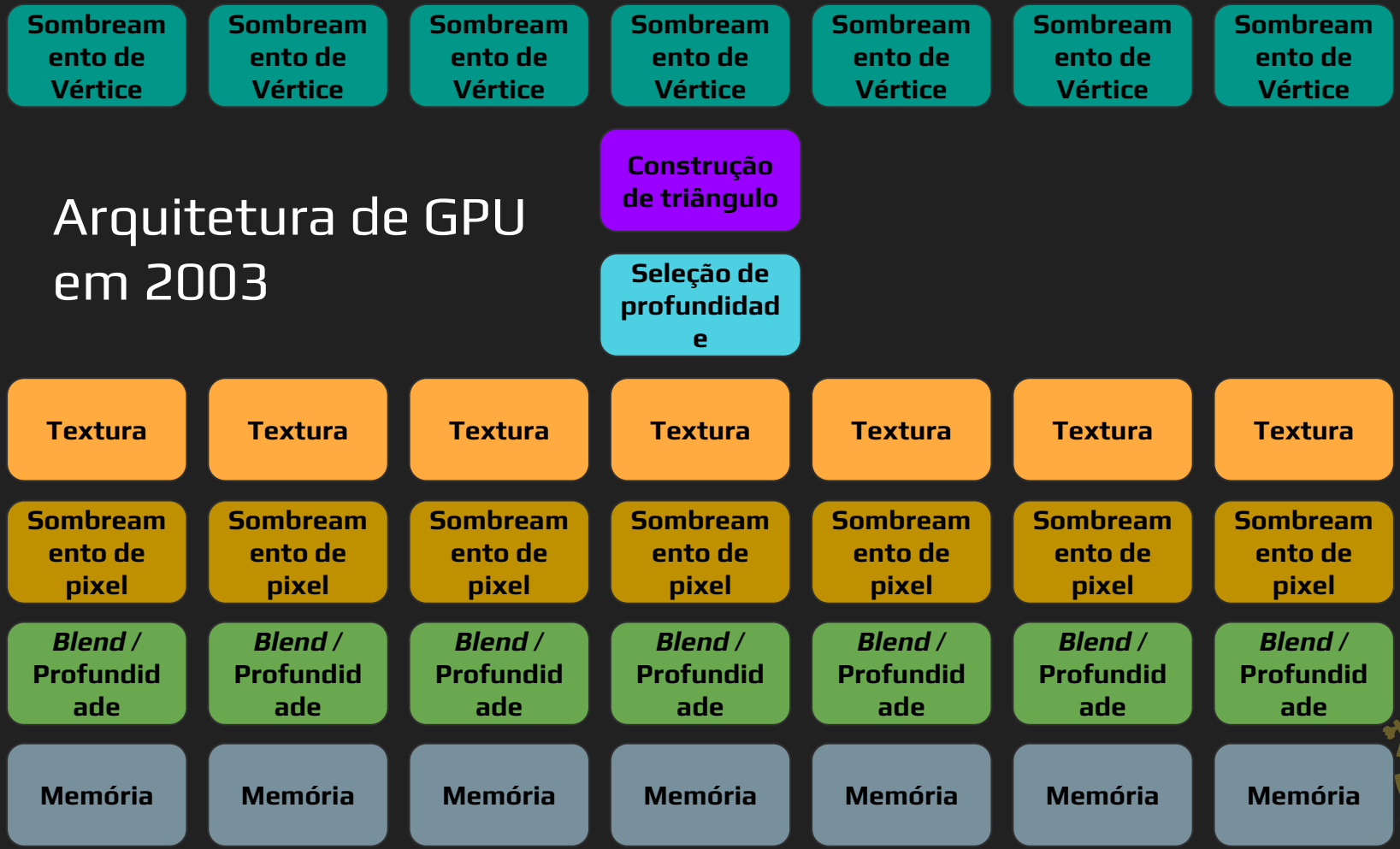
5. *Pipeline & Hardware*

- *Pipeline*:
  - Encadeamento de comandos
  - Ordem na qual os comandos são executados
- Antigamente:
  - *Chips*, placas e/ou unidades distintas por estágio
  - Fluxo de dados fixo pelo *pipeline*
  - *Hardware* costumava seguir isso:

Pipeline


Transformada de vértices e iluminação
Construção de triângulos e rasterização
Texturização e sombreamento de pixel
Teste de profundidade e *blending* (composição)
Transformada de vértices e iluminação

Arquitetura de GPU em 2003



5. *Pipeline & Hardware*

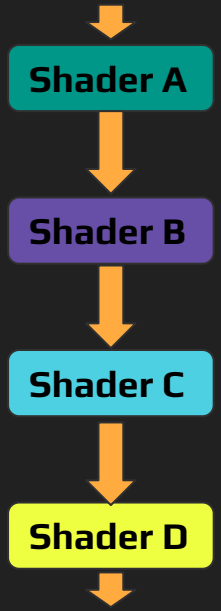
- Atualmente
  - GPUs totalmente programáveis
  - Arquitetura unificada
  - Nova funcionalidade - *geometry shader*
  - Programável em C
  - Fluxo de dados arbitrário
  - Modelo de programação de múltiplos propósitos

5. *Pipeline & Hardware*

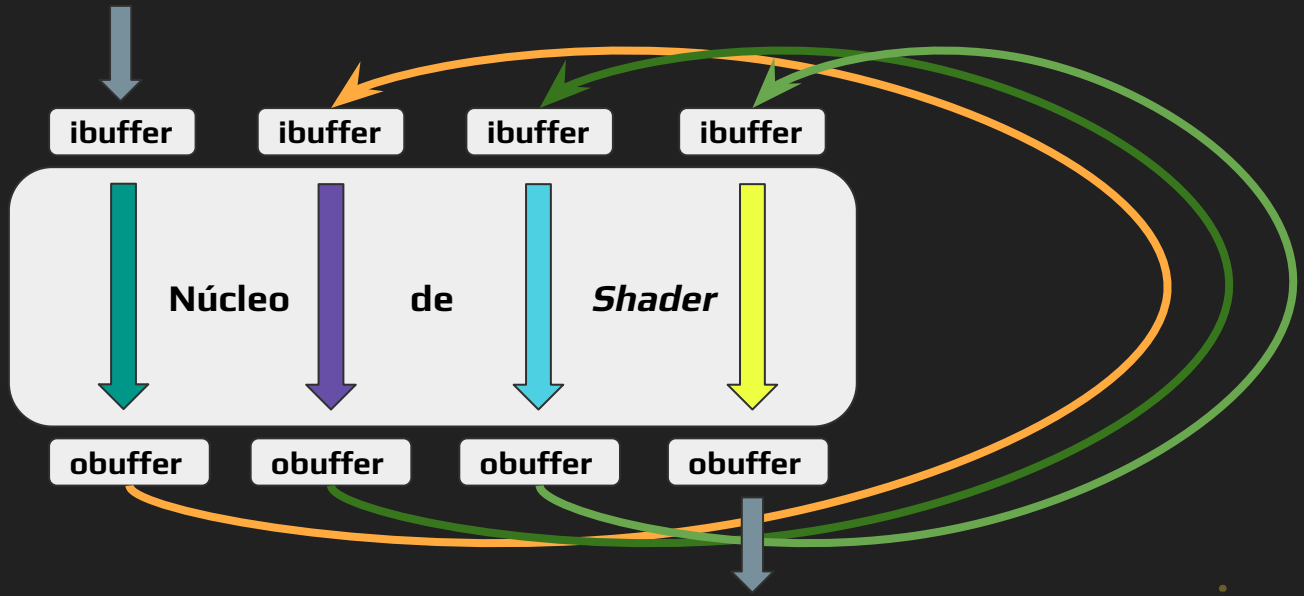
- Atualmente
  - *Hardware* de propósito específico
  - Threading e pipelining gerenciados por *hardware*
  - “Gráficos” e computação livremente misturados

Design discreto X unificado

Design discreto

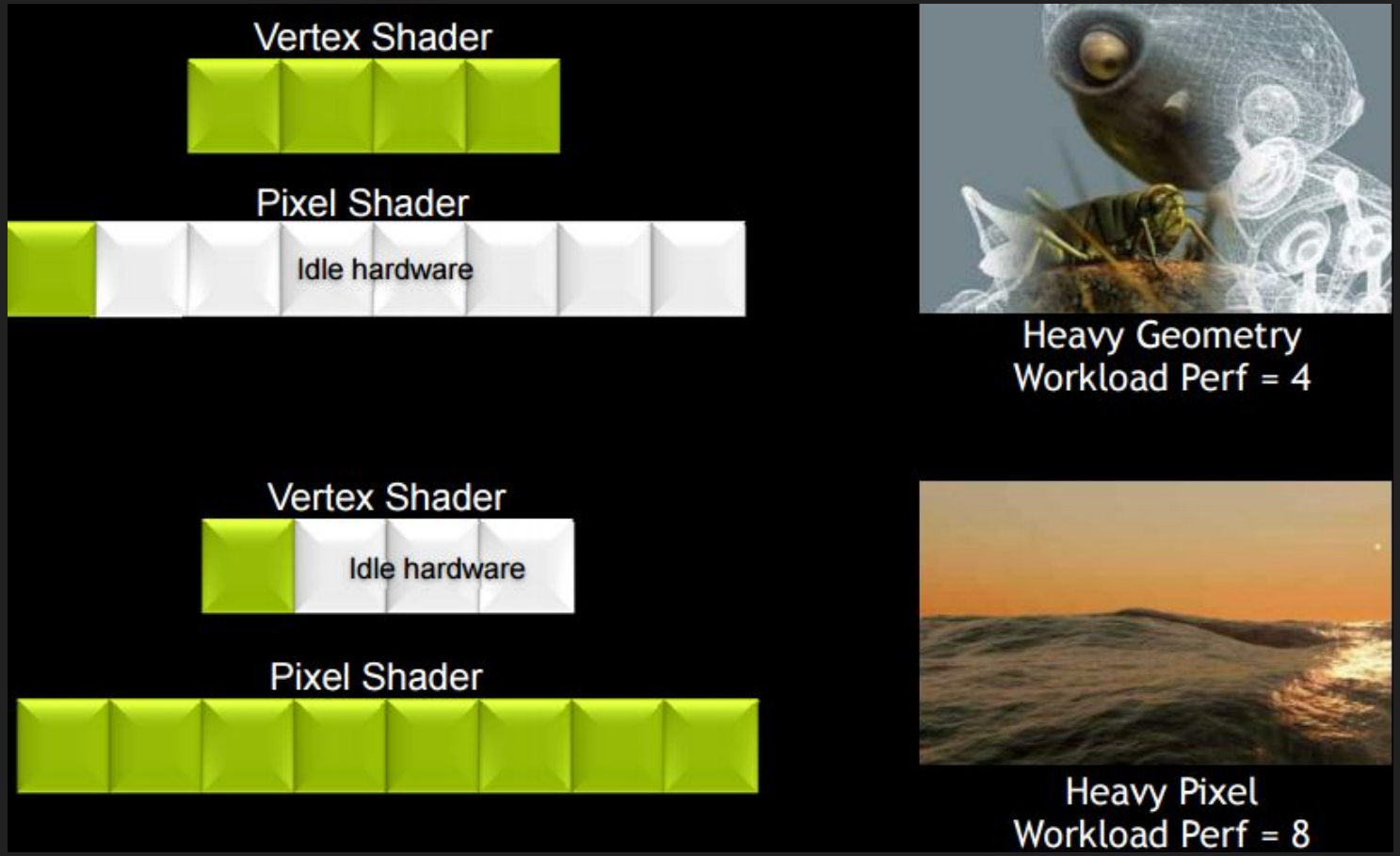


Design unificado



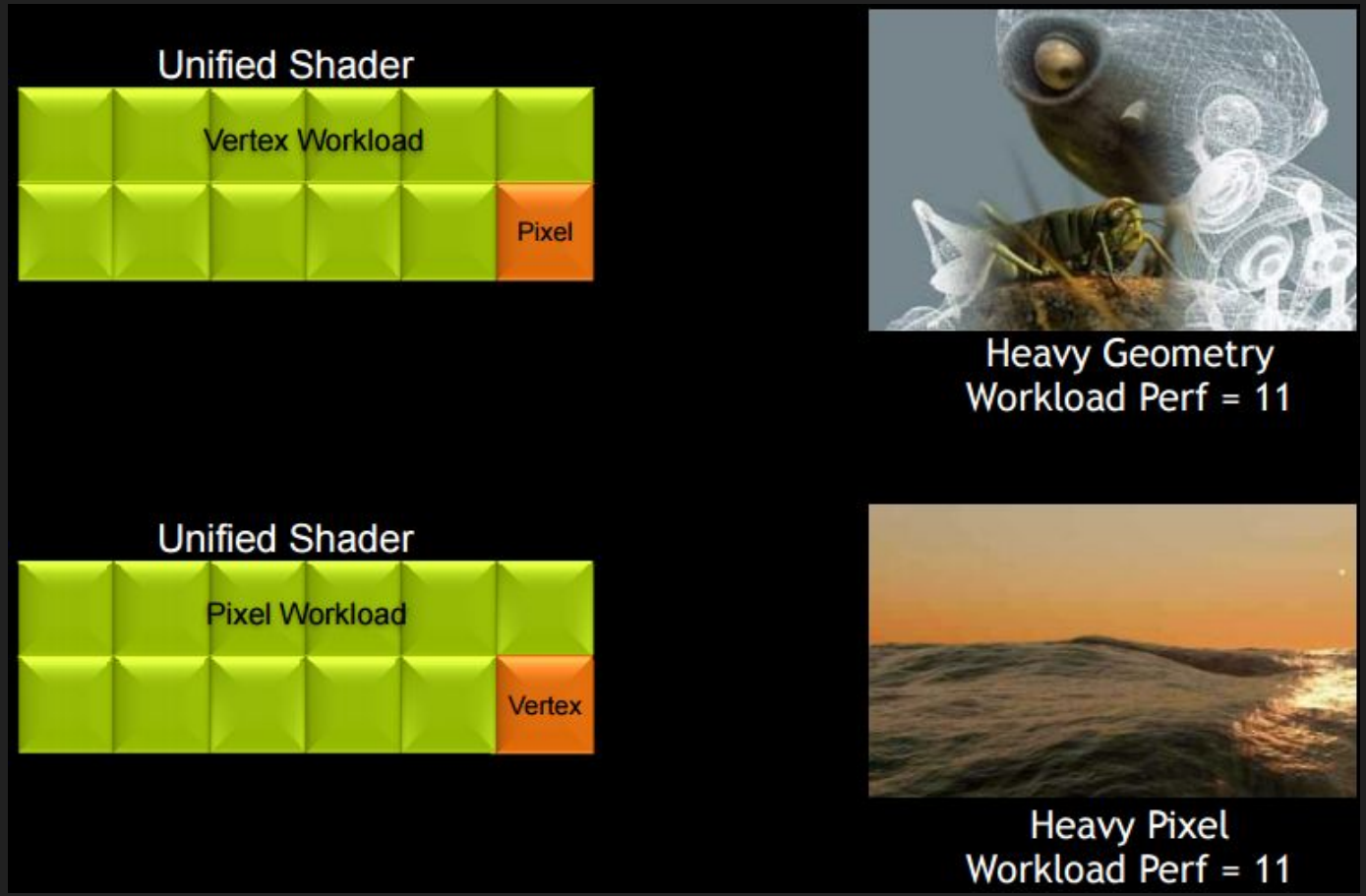
Arquitetura unificada: Sombreamentos de vértices, pixels, etc. tornam-se *threads* rodando em diferentes programas em núcleos flexíveis

Por que unificar?



http://www.nvidia.com/content/nvision2008/tech_presentations/Technology_Keynotes/NVISION08-Tech_Keynote-GPU.pdf

Por que unificar?

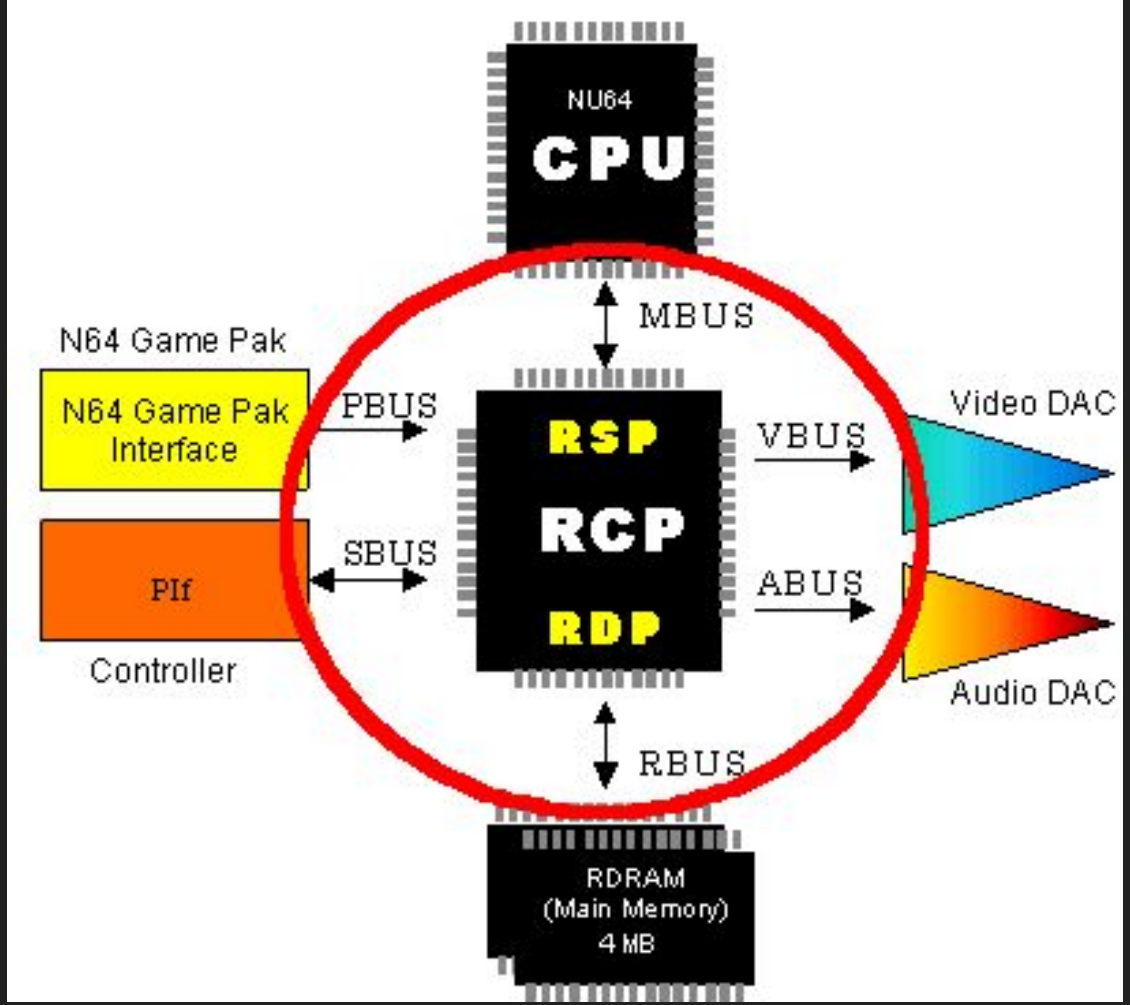


http://www.nvidia.com/content/nvision2008/tech_presentations/Technology_Keynotes/NVISION08-Tech_Keynote-GPU.pdf

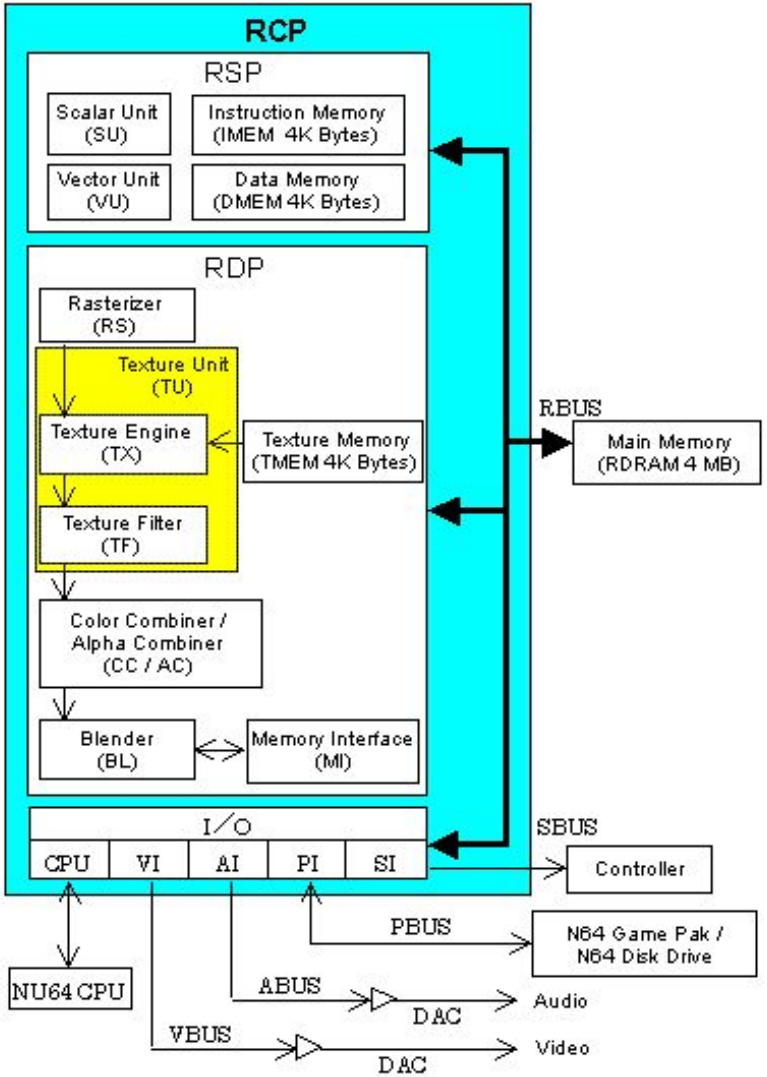
Bonus Stage 3: Arquitetura do N64

Bonus Stage 3: Arquitetura do N64

- ➔ Como funcionava o pipeline do Nintendo 64?
  - ◆ Reality Co-Processor (RCP) é o componente mais importante no N64
    - Todos os dados passam por ele
    - E serve como um controlador de memória da CPU
  - ◆ O RCP tem 2 processadores (RSP e RDP)
  - ◆ E interfaces de I/O (VI, AI, PI e SI)



http://n63.icequake.net/doc/n64intro/kantan/images/1-2-3-1.gif



Blocos de processos do RCP

http://n63.icequake.net/doc/n64intro/kantan/images/1-2-3-2.gif

Bonus Stage 3: Arquitetura do N64

- Microcódigo
  - Técnica que impõe um interpretador entre o *hardware* e o nível de arquitetura de um computador
  - Camada de instruções a nível de *hardware* que implementam instruções de código de máquina de nível maior ou sequenciamento de máquinas de estado internas em vários elementos de processamento digital

Bonus Stage 3: Arquitetura do N64

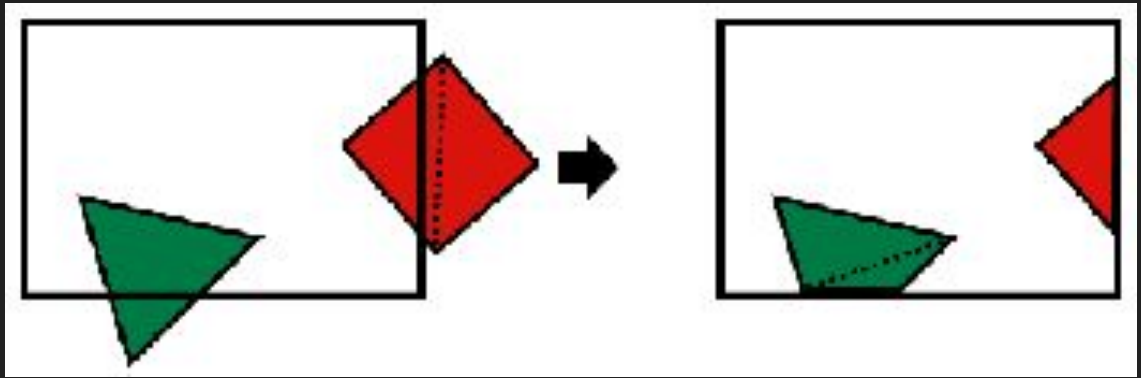
- Reality Signal Processor (RSP)
  - Executa tarefas de áudio e vídeo
  - Usando microcódigo completa as seguintes tarefas
    - Operações na pilha de matriz
    - Transformações geométricas 3D
    - *Frustrum culling* e *back-face rejection* (*culling*)
    - Mapeamento de iluminação e reflexão
    - Construção da rasterização de linha e polígonos

Bonus Stage 3: Arquitetura do N64

- ➔ Processos gráficos executados no RSP
  - ◆ A maioria dos processos do RSP são executados quando os dados de vértices são carregados no *cache* de vértices
  - ◆ Os processos principais são:
    - Transformadas geométricas
      - Necessárias para mover ou escalar objetos 3D
    - RSP faz todas as transformadas necessárias

Bonus Stage 3: Arquitetura do N64

- Processos gráficos executados no RSP
  - *Clipping* (recorte)
    - Recorta polígonos ou pedaços fora da visão da tela



http://n63.icequake.net/doc/n64intro/kantan/images/1-2-3-5.gif

Bonus Stage 3: Arquitetura do N64

- ➔ Processos gráficos executados no RSP
  - ◆ *Culling* (Seleção)
    - Seleciona dados desnecessários para o *pipeline* gráficos
    - Exemplo: dados para desenhar a parte “de trás” de um objeto
    - N64 suporta dois tipos de *culling*:
      - *Back-face* e de volume

Bonus Stage 3: Arquitetura do N64

- ➔ Back-face culling
  - ◆ Retira a parte de trás de objetos

http://n63.icequake.net/doc/n64intro/kantan/images/1-2-3-6.gif

Bonus Stage 3: Arquitetura do N64

- Volume culling
  - Retira itens que estão completamente fora do campo visual da lista de exibição (*display list*) que desenha objetos

http://n63.icequake.net/doc/n64intro/kantan/images/1-2-3-7.gif

Bonus Stage 3: Arquitetura do N64

- ➔ Processos gráficos executados no RSP
  - ◆ Cálculos de iluminação:
    - Calcula iluminação
      - Falaremos mais deste processo depois!

Bonus Stage 3: Arquitetura do N64

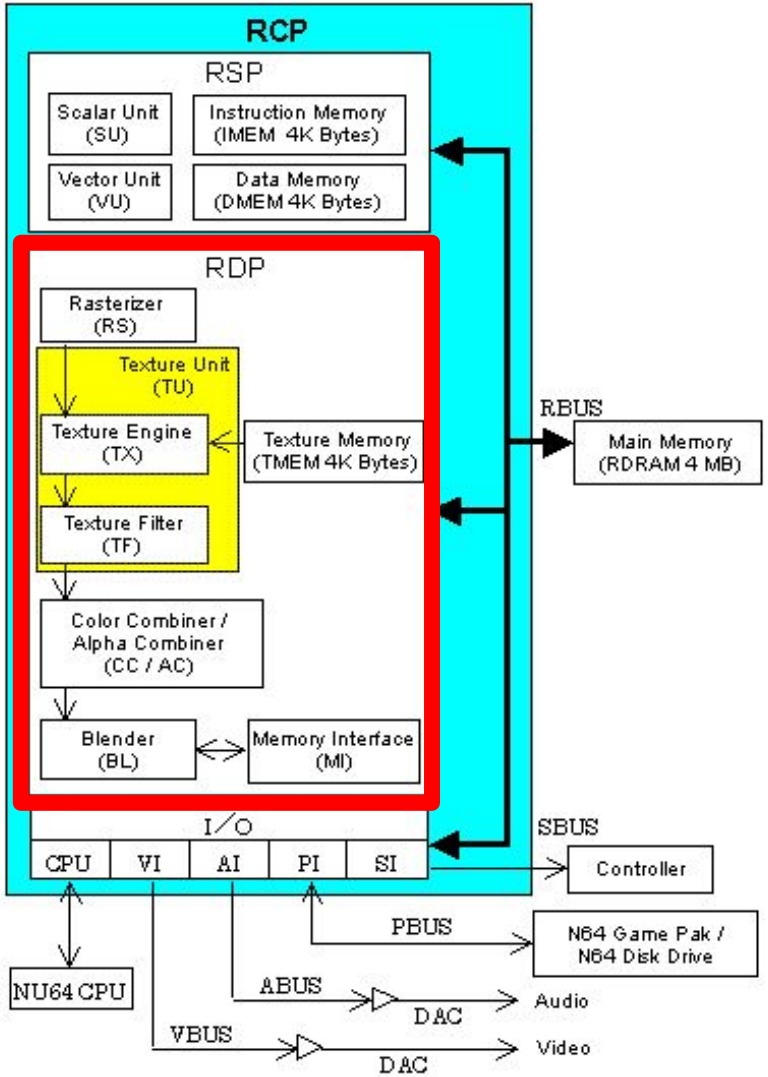
- ➔ Reality Display Processor (RDP)
  - ◆ Processa a *display list* criada pelo RSP e CPU para criar os dados gráficos.
  - ◆ Rasteriza triângulos e retângulos
  - ◆ Produz pixels de alta qualidade que são
    - Texturizados
    - Suavizados (anti-aliased)
    - Colocados no z-*buffer*

Bonus Stage 3: Arquitetura do N64

- ➔ Reality Display Processor (RDP)
  - ◆ Trabalha apenas com gráficos
  - ◆ Desenha os gráficos do *frame buffer*
  - ◆ Processa diversas operações relacionadas ao desenho

Bonus Stage 3: Arquitetura do N64

- Reality Display Processor (RDP) operações:
  - Rasterização de polígonos
  - Texturização e aplicação de filtros
  - Combinação (alfa + cores)
  - *Z-buffering*
  - *Anti-aliasing*



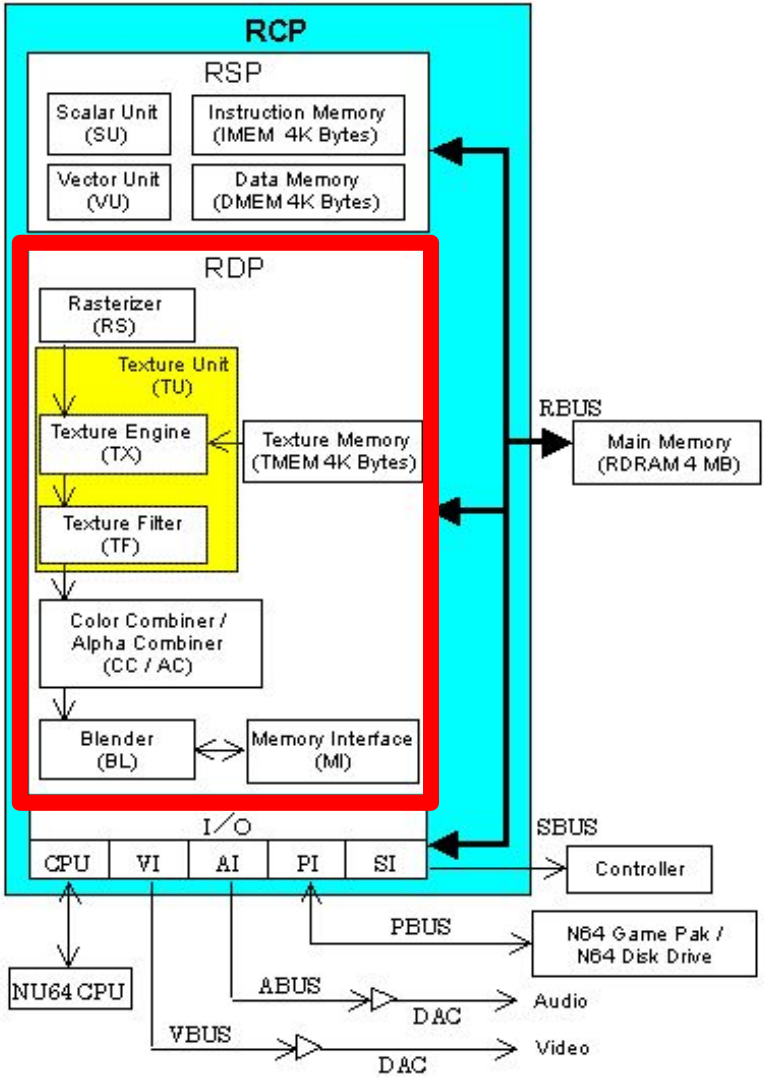
Unidades do RDP

- Rasterizador (RS) transforma triângulos e retângulos em pixels
- Motor de Textura (TX) provê amostragem para texels (elementos de imagem) através do uso de TMEM (Memória de Textura)
- Filtro de Textura (TF) provê filtragem para texels criados pelo TX
- Combinador de Cor/Combinador Alpha (CC/AC) combina duas cores de pixels criadas pelo RS e texels criados pelo TF e interpola entre essas duas cores.

http://n63.icequake.net/doc/n64intro/kantan/images/1-2-3-2.gif

Unidades do RDP

- O Misturador (Blender) (BL) mistura a cor do pixel determinada em CC, a cor no *frame buffer*, a cor de *fog*, e assim em diante. Desenha a cor resultante no *frame buffer*. Nestemomento, também provê o Z-*buffering* para a primeira parte do processo de *anti-aliasing*.
- Interface de Memória (MI) processa informação de pixel no *frame buffer* incluindo operações de ler, modificar e escrever

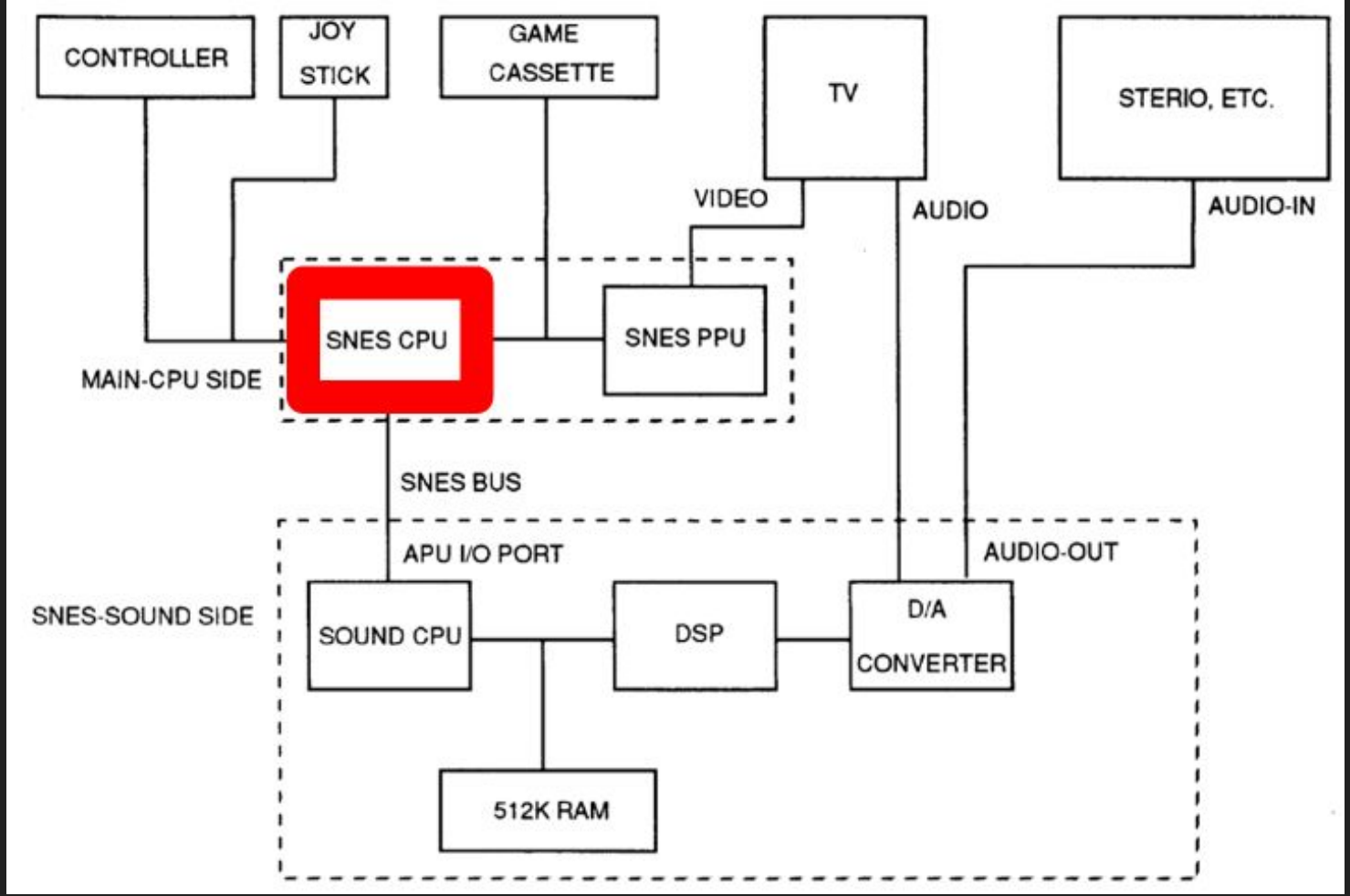


http://n63.icequake.net/doc/n64intro/kantan/images/1-2-3-2.gif

Bonus Stage 4: Arquitetura do SNES

Bonus Stage 4: Arquitetura do SNES

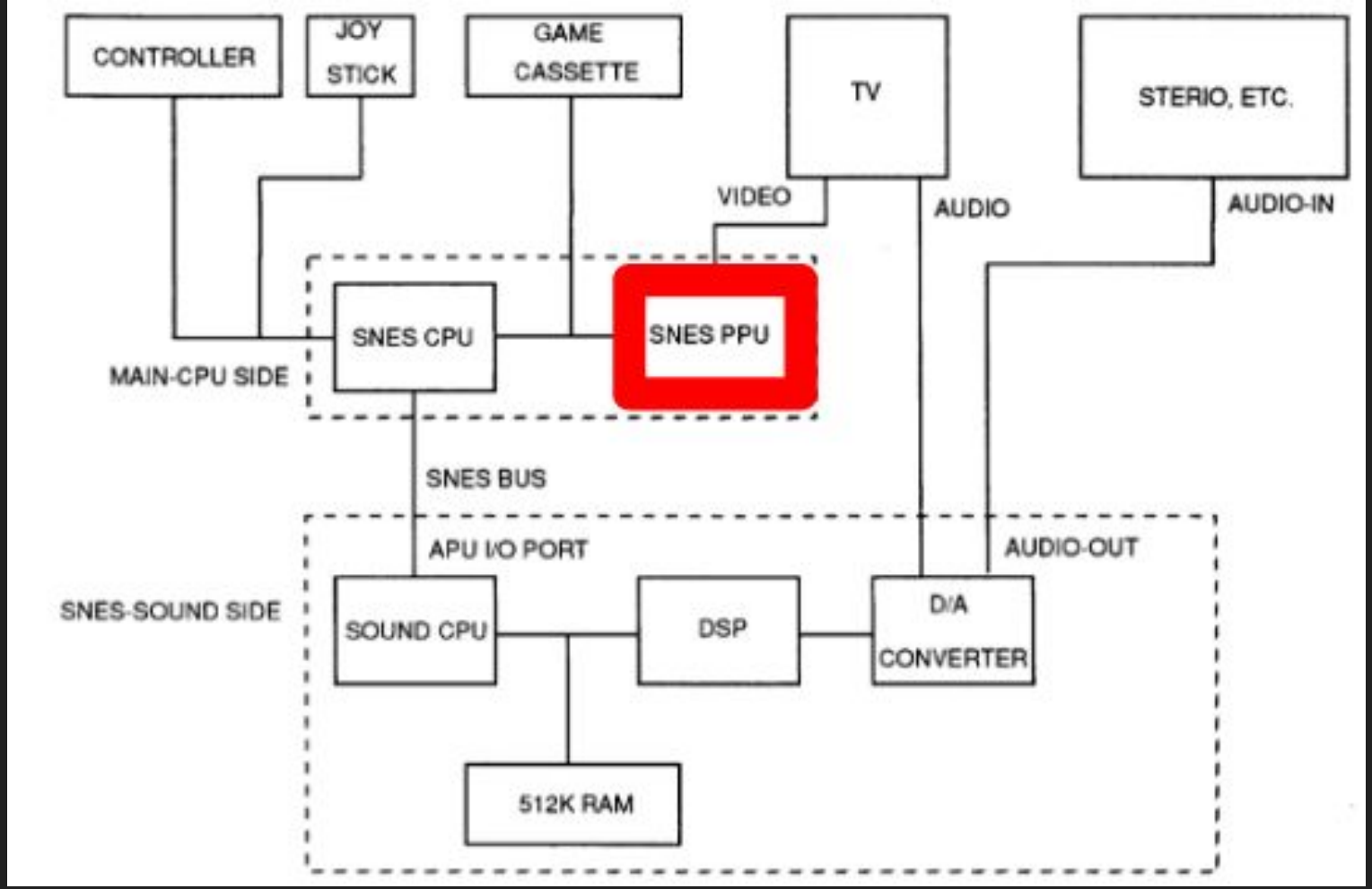
- SNES
  - Velocidade da CPU 2.86MHz (até 10.74 MHz)
  - 16bits
  - 24 bit bus - usado para acessos gerais
  - 8 bit bus - usado para acesoss de registro APU e PPU
  - Instruções por segundo: 1.79 MIPS



http://meseec.ce.rit.edu/551-projects/fall2014/3-1.pdf

Bonus Stage 4: Arquitetura do SNES

- SNES
  - Velocidade da CPU 2.86MHz (até 10.74 MHz)
  - 16bits
  - 24 bit bus - usado para acessos gerais
  - 8 bit bus - usado para acesoss de registro APU e PPU
  - Instruções por segundo: 1.79 MIPS



http://meseec.ce.rit.edu/551-projects/fall2014/3-1.pdf

Bonus Stage 4: Arquitetura do SNES

- Picture Procesing Unit (PPU)
  - 2 unidades: PPU1 e PPU2
  - PPU1 gera dados, rotação e escala de caracteres de background
  - PPU2 realiza efeitos especiais
  - 32.768 cores
  - Mesmo sinal de clock da CPU
  - 7 modos de vídeo diferente

Bonus Stage 4: Arquitetura do SNES

- Picture Procesing Unit (PPU)
  - 64kB de RAM
  - 256x224
  - 512x224
  - 256x239
  - 512x239
  - 512x448
  - 512x478

Bonus Stage 4: Arquitetura do SNES

- Paleta de cores
  - Formato de BGR 15-Bit
  - 512 bytes para a paleta, cada palavra com 2 bytes
    - 256 cores
    - Cada valor de cor varia de 0 a 31
    - 31, 31, 31 é branco
    - Diferente do padrão 24-bit RGB, que varia de 0 a 255

Bonus Stage 4: Arquitetura do SNES

- SNES pode mostrar 256 cores de uma vez
  - Divididas em 16 sub-paletas, com 16 cores (1 sempre transparente)
  - *Tiles* de BG usam qualquer uma das 8 primeiras sub-paletas, enquanto os *sprites* usam as outras 8

Bonus Stage 4: Arquitetura do SNES

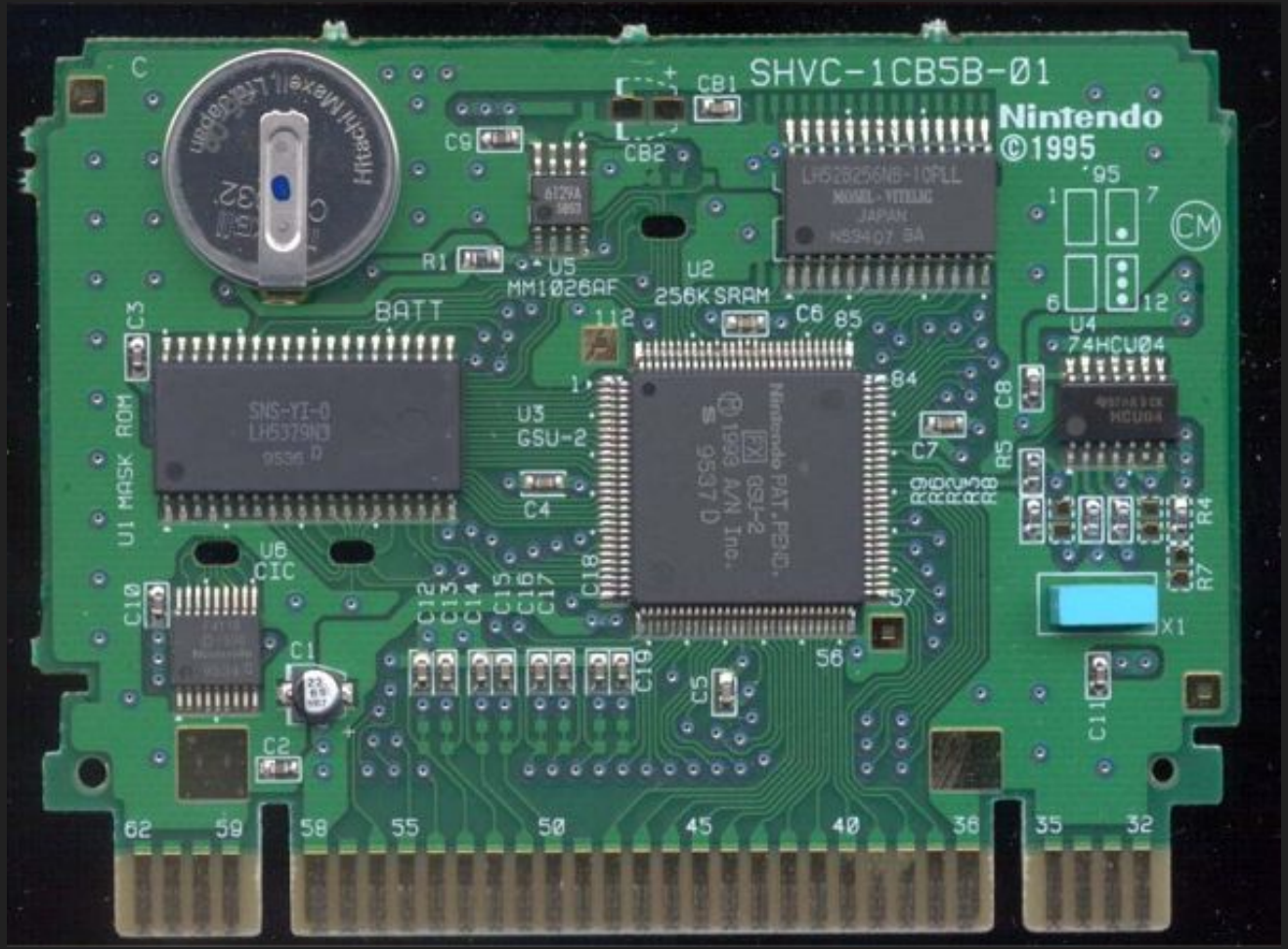
- Modos de vídeo
  - Alteram entre si quantidade de camadas e cores nas paletas
  - Especificação e decodificação de elementos varia

Bonus Stage 4: Arquitetura do SNES

- Mode 7
  - Camada única que pode ser rotacionada e escalada usando transformações de matrizes
  - HDMA (Horizontal Direct Memory Access) é normalmente usado para mudar os parâmetros da matriz para cada *scanline* para gerar efeitos de perspectiva

Bonus Stage 4: Arquitetura do SNES

- Cartuchos
  - Super FX chip
    - CPU RISC
    - Renderiza gráficos que a CPU normal não consegue
    - Processa principalmente polígonos 3D
    - 10.5 MHz de clock
  - Super Accelerator System



http://meseec.ce.rit.edu/551-projects/fall2014/3-1.pdf

Bonus Stage 4: Arquitetura do SNES

- Cartuchos
  - Super Accelerator System (SA-1)
    - CPU melhorada colocada no cartucho
    - Processador 16-bit
    - 10.74 MHz de clock
    - Trabalha em paralelo com processador original
      - 5x a performance original

Bonus Stage 4: Arquitetura do SNES

- ➔ Cartuchos
  - ◆ Super Accelerator System (SA-1)
    - 2kB de RAM interna, 2MB de RAM externa
    - 64MB de ROM externa

Engine do DKC 2 (feita em Assembly)

Dúvidas?

Referências

Referências

[1]http://www.nvidia.com/content/nvision2008/tech_presentations/Technology_Keynotes/NVISION08-Tech_Keynote-GPU.pdf
[2]http://n64.icequake.net/doc/n64intro/kantan/
[3]https://en.wikipedia.org/wiki/Microcode
[4]https://en.wikipedia.org/wiki/Texel_(graphics)
[5]http://www.nvidia.com/object/cuda_home_new.html
[6]https://llpanorama.wordpress.com/2008/05/21/my-first-cuda-program/
[7]http://www.nvidia.com/content/gtc/documents/1055_gtc09.pdf
[8]https://en.wikipedia.org/wiki/Computer_graphics
[9]http://www.graphics.cornell.edu/online/tutorial/
[10]https://en.wikipedia.org/wiki/2D_computer_graphics
[11]https://en.wikipedia.org/wiki/Pixel_art
[12]https://en.wikipedia.org/wiki/Font_rasterization
[13]http://acko.net/files/fullfrontal/fullfrontal/webglmath/online.html
[14]http://blog.digitaltutors.com/bump-normal-and-displacement-maps/

Referências

[12]https://en.wikipedia.org/wiki/Sprite_(computer_graphics)
[13]https://en.wikipedia.org/wiki/Vector_graphics
[14]https://en.wikipedia.org/wiki/3D_computer_graphics
[15]https://en.wikipedia.org/wiki/Computer_animation
[16]https://en.wikipedia.org/wiki/Uncanny_valley
[17]https://www.youtube.com/watch?v=eN3PsU_iA80
[18]https://images.nvidia.com/content/pdf/tesla/whitepaper/pascal-architecture-whitepaper.pdf
[19]http://meseec.ce.rit.edu/551-projects/fall2014/3-1.pdf

Referências Complementares

[1]http://nesdev.com/NESDoc.pdf
[2]http://www.vasulka.org/archive/Writings/VideogameImpact.pdf#page=24
[3]https://en.wikipedia.org/wiki/Real-time_computer_graphics
[4]http://dkc-forever.blogspot.com.br/2015/11/curiosidades-designer-da-rare-revela.html
[5]http://level42.ca/projects/ultra64/Documentation/man/