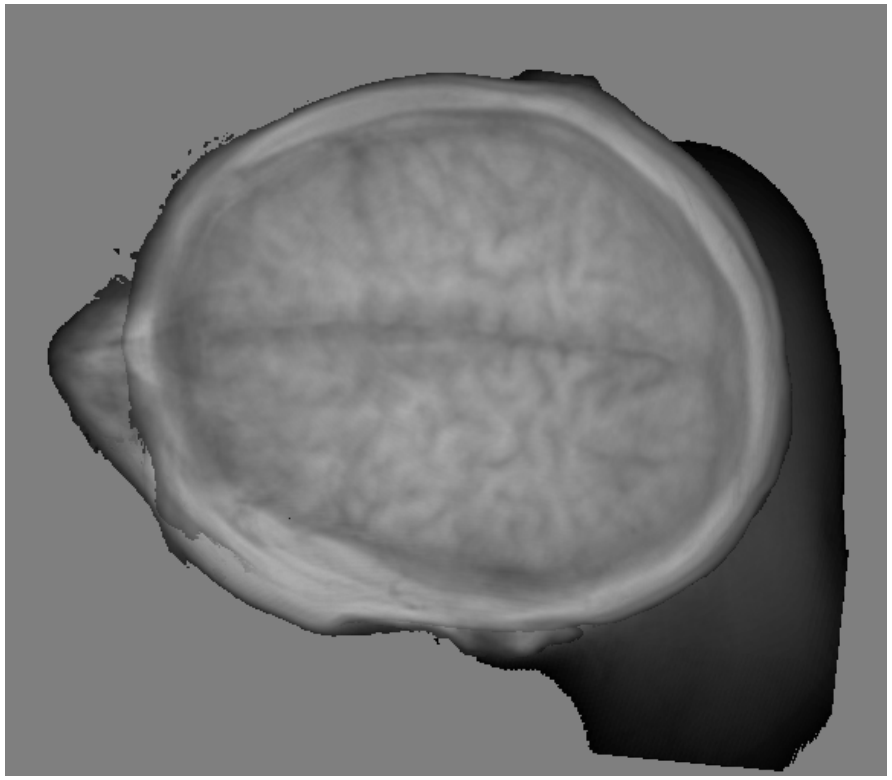
Uma Introdução à Visualização Volumétrica para Diagnóstico de Lesões Cerebrais

Módulo II



Wu Shin Ting
ting@dca.fee.unicamp.br
FEEC - Unicamp

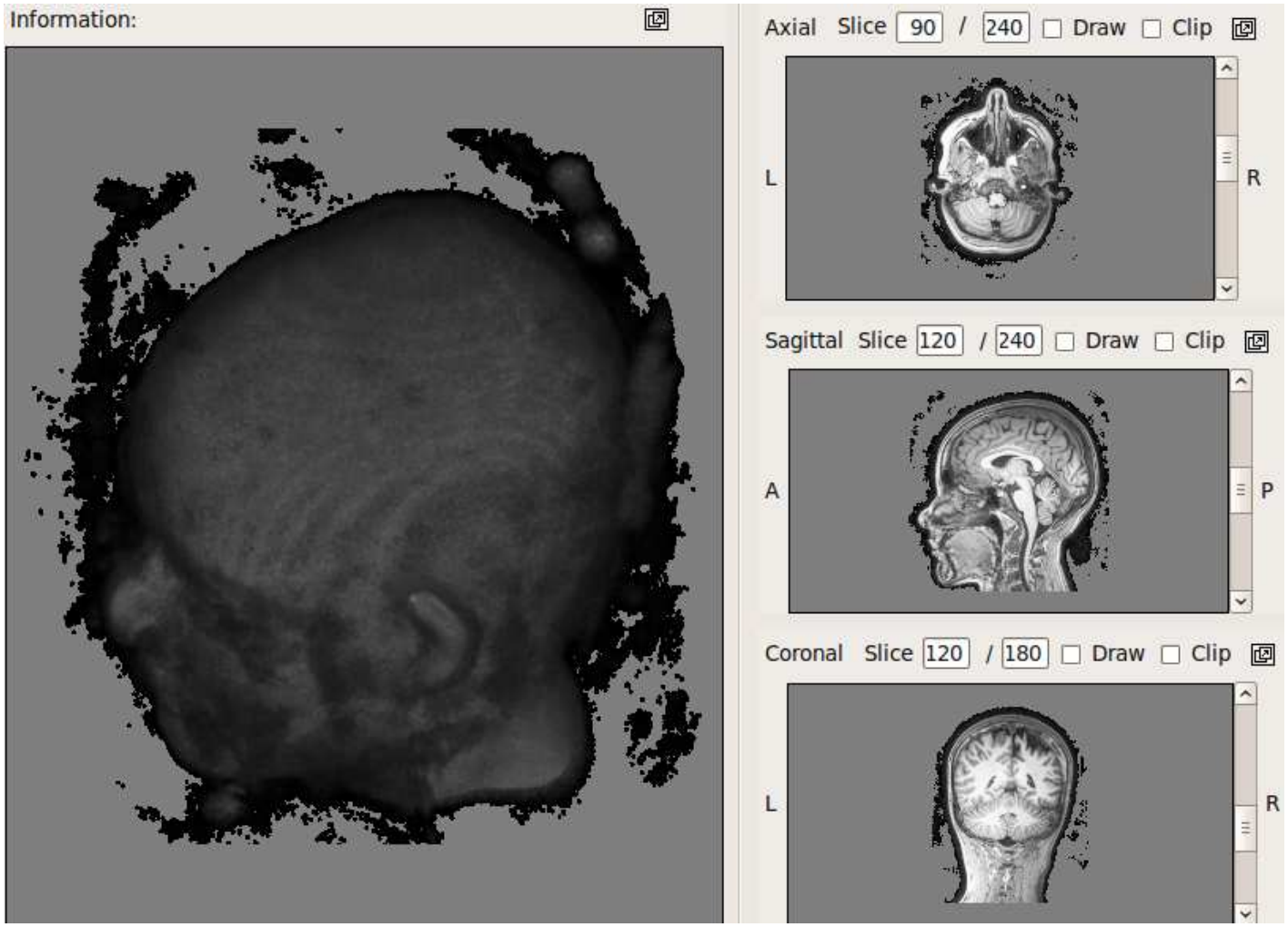
Clarissa Lin Yasuda
yasuda.clarissa@gmail.com
FCM - Unicamp

Organização do Minicurso

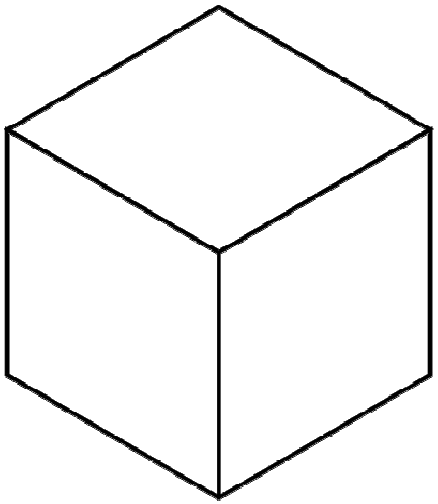
- Protocolo de diagnóstico de displasia cortical focal
- Modelo matemático de neuroimagens e técnicas de renderização
- Tecnologia de renderização: GPUs
- Estado-de-arte de visualização de neuroimagens
- Implementação de ferramentas de interação

Técnicas de Renderização: MPR

- Reformatação Multiplanar

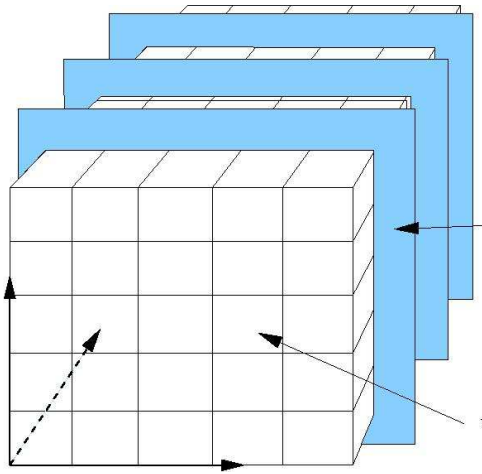


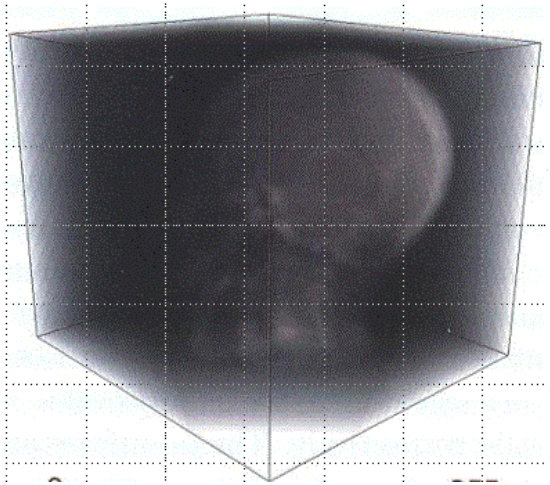
Técnicas de Renderização: Texturização 3D

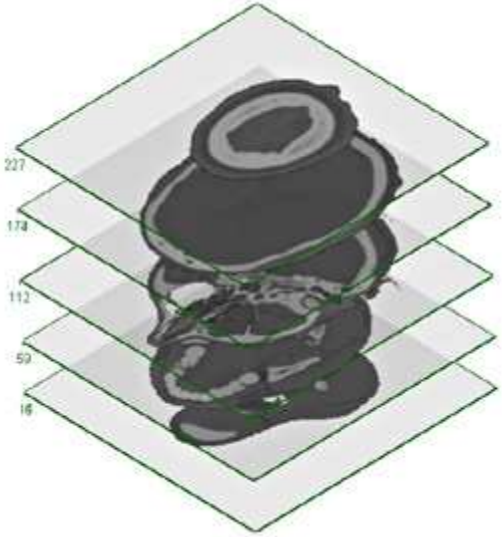


Proxy Geometry

Espaço do objeto







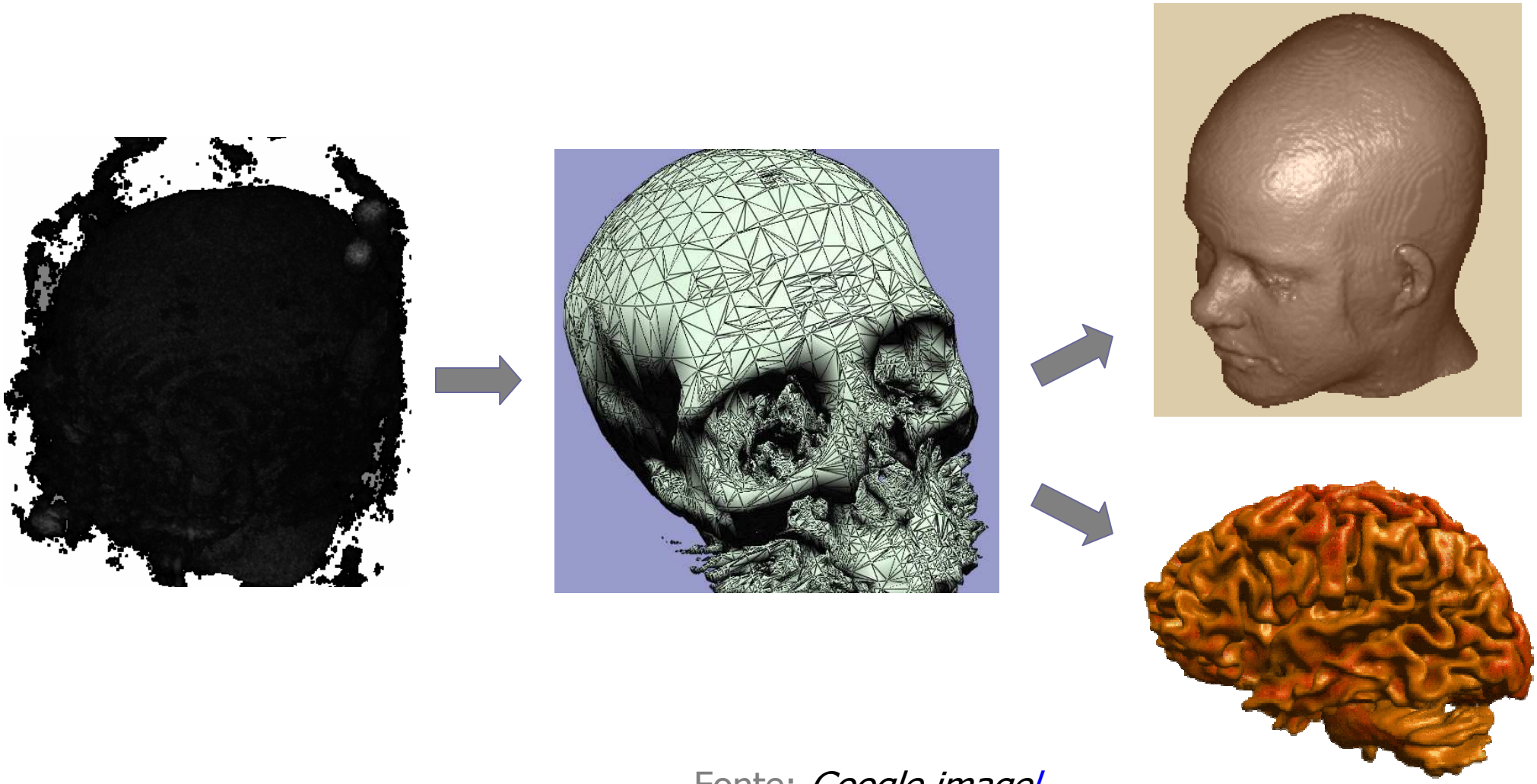
Volume adquirido

Espaço de textura

Imagem Final

Técnicas de Renderização: ISR

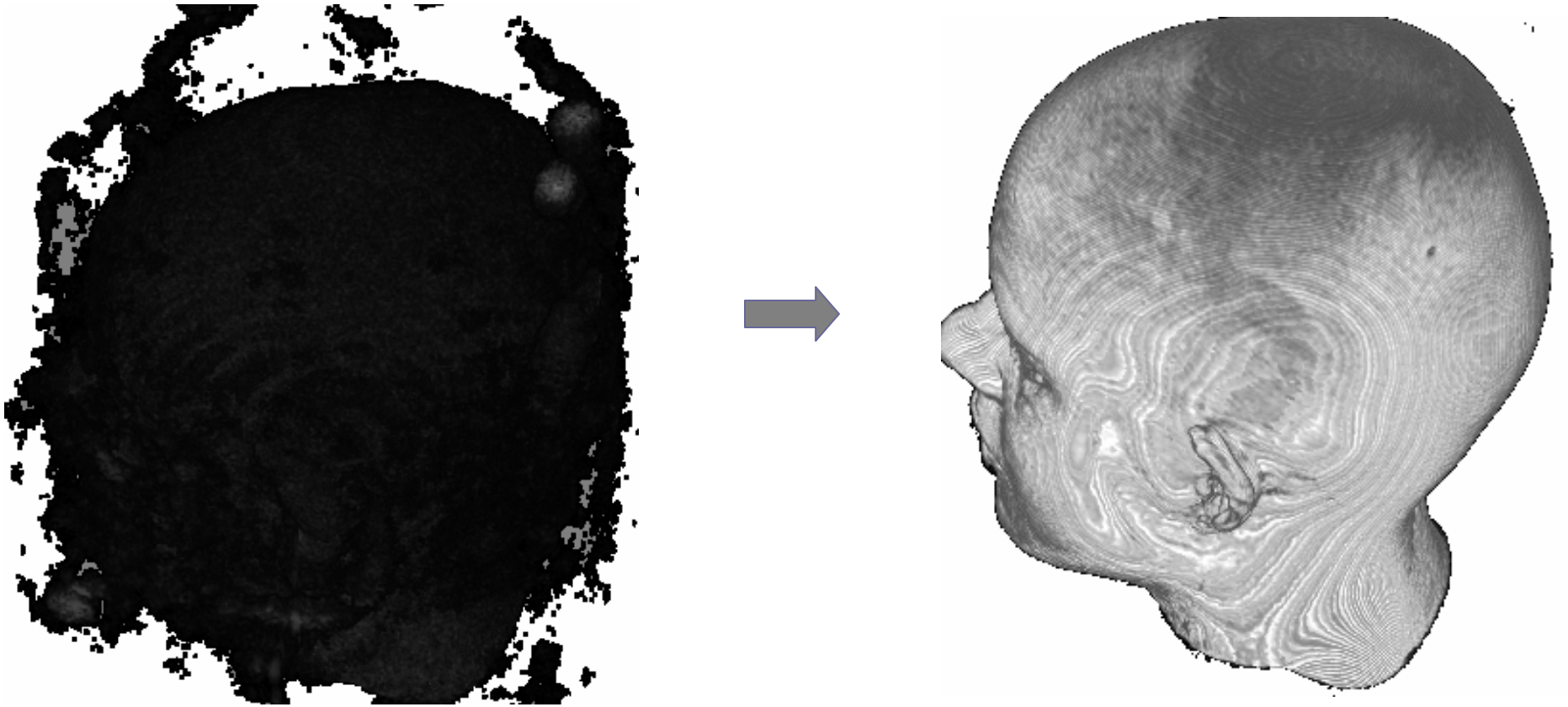
- Renderização Indireta



Fonte: *Google image*

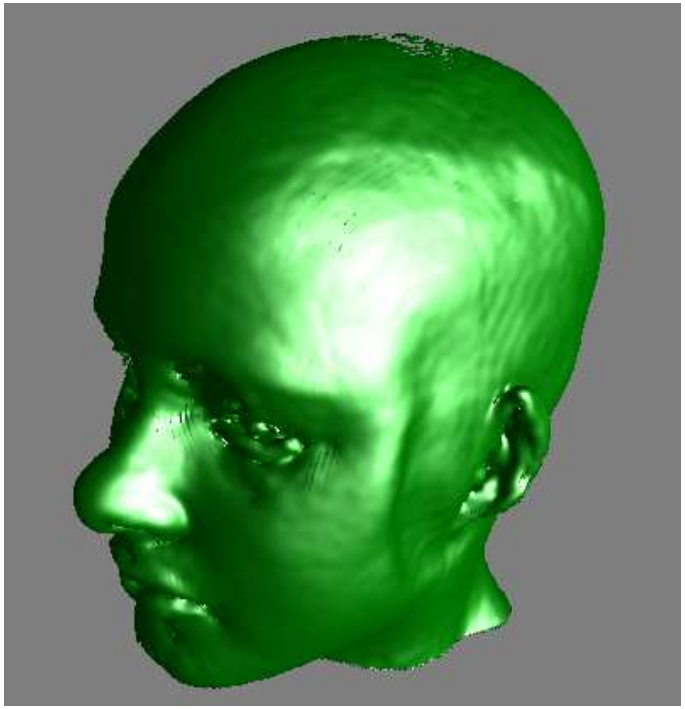
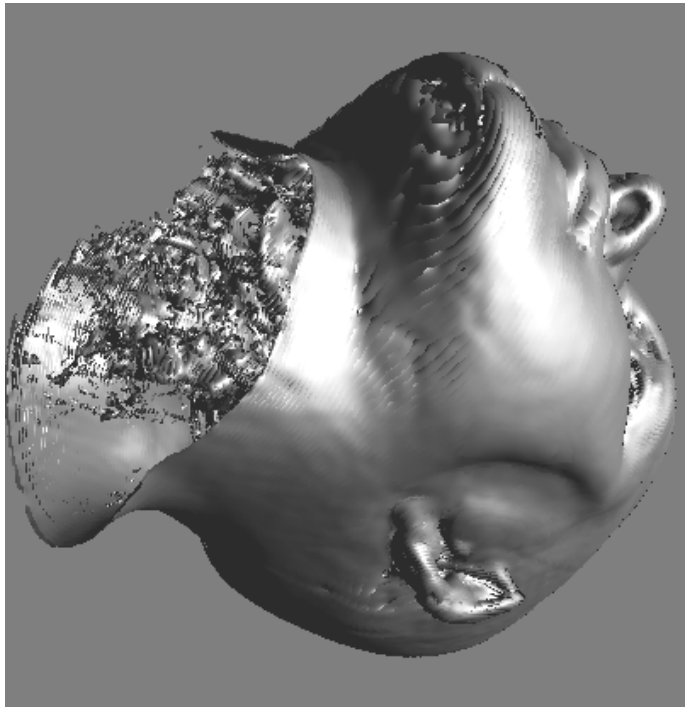
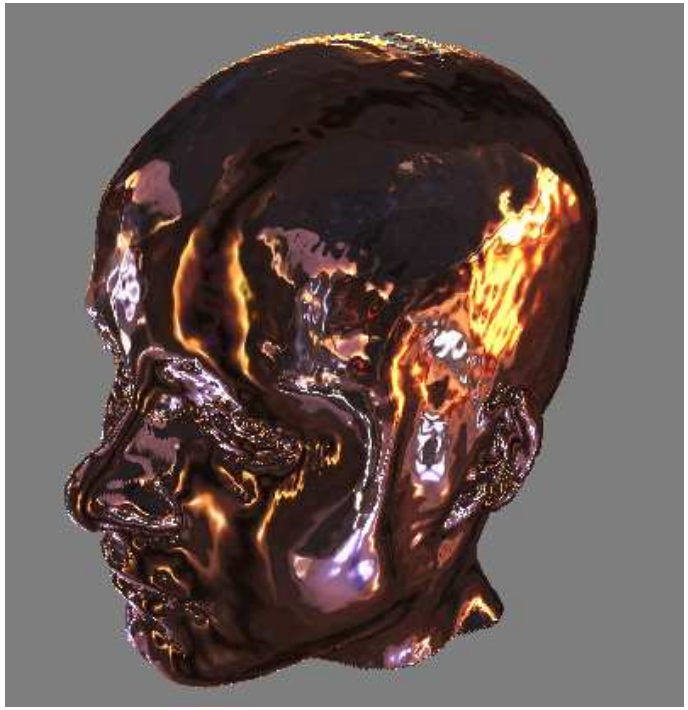
Técnicas de Renderização: DR

- Renderização Direta
  - *Direct Surface Rendering* (DSR)
  - *Direct Volume Rendering* (DVR)



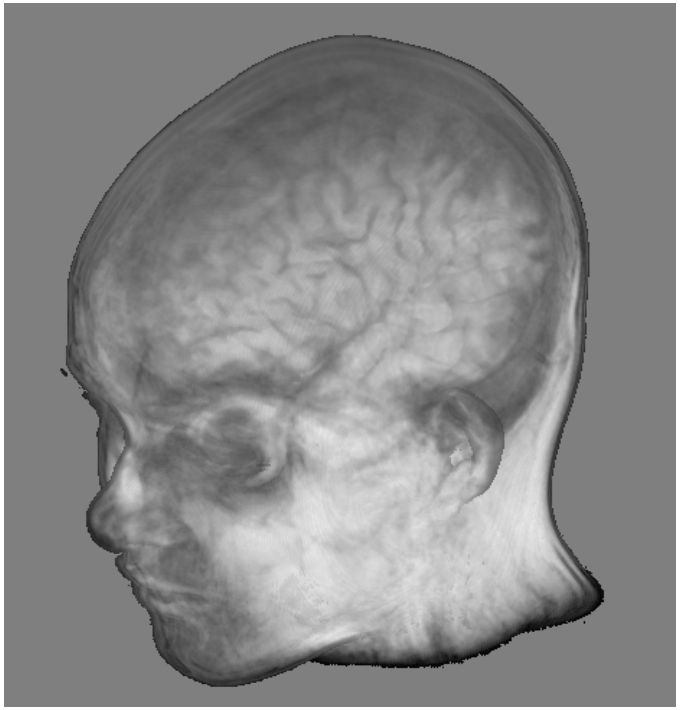
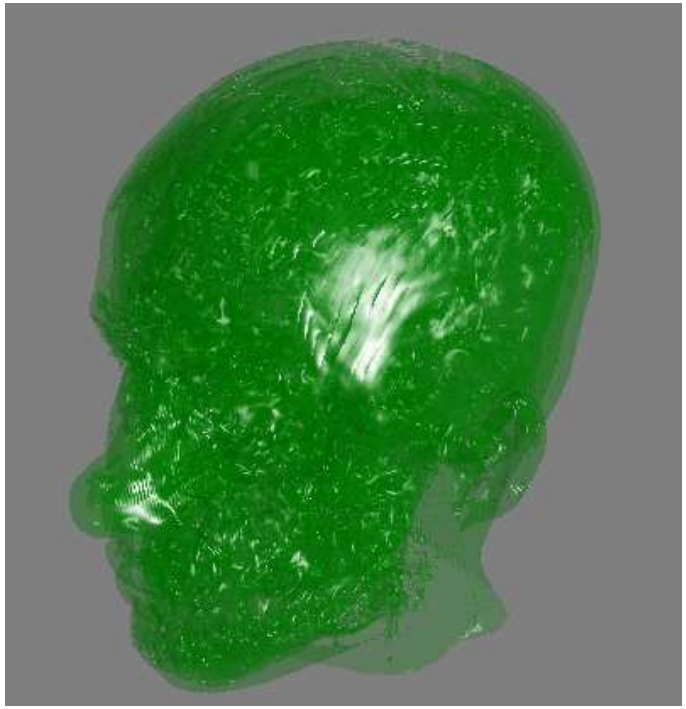
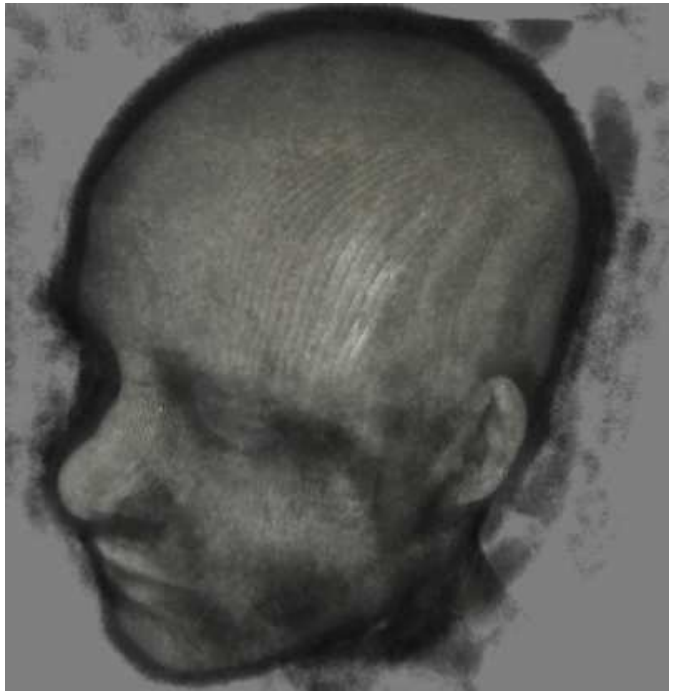
Raycasting: DSR

- Modelo de iluminação clássica em amostras visíveis
  - Gradientes das intensidades → vetor normal

***Raycasting*: DVR**

- Composição das intensidades
  - intensidades → cor e opacidade

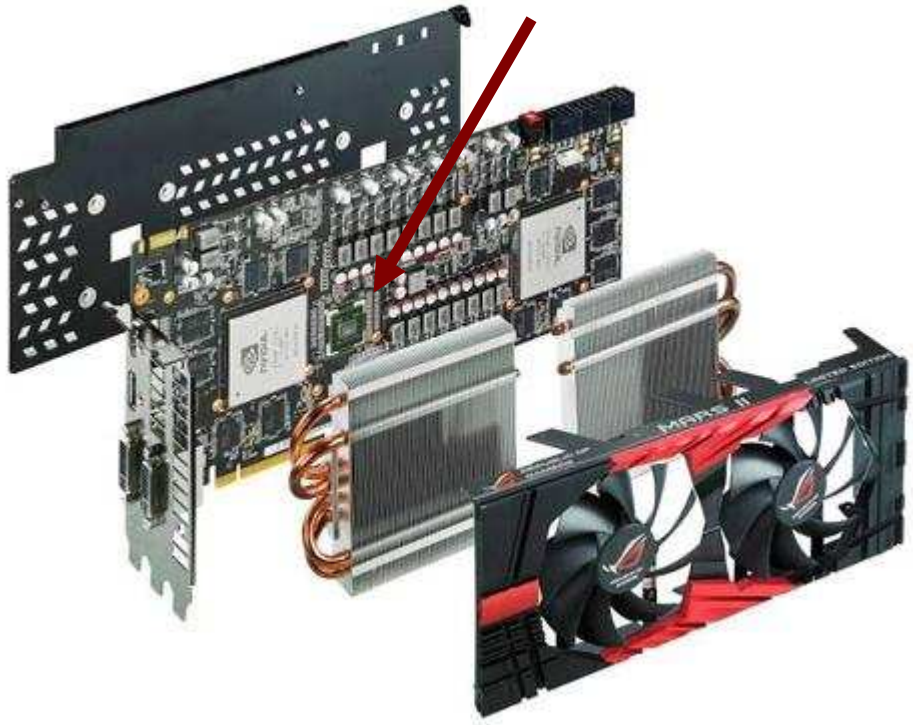




Organização do Minicurso

- Protocolo de diagnóstico de displasia cortical focal
- Modelo matemático de neuroimagens e técnicas de renderização
- Tecnologia de renderização: GPUs
- Estado-de-arte de visualização de neuromagens
- Implementação de ferramentas de interação

Unidades de Processamento Gráfico (GPUs)

- Processador dedicado para renderização em tempo real.
- *Onboard* ou *offboard*.
- Três maiores fabricantes: nVidia, ATI e Intel

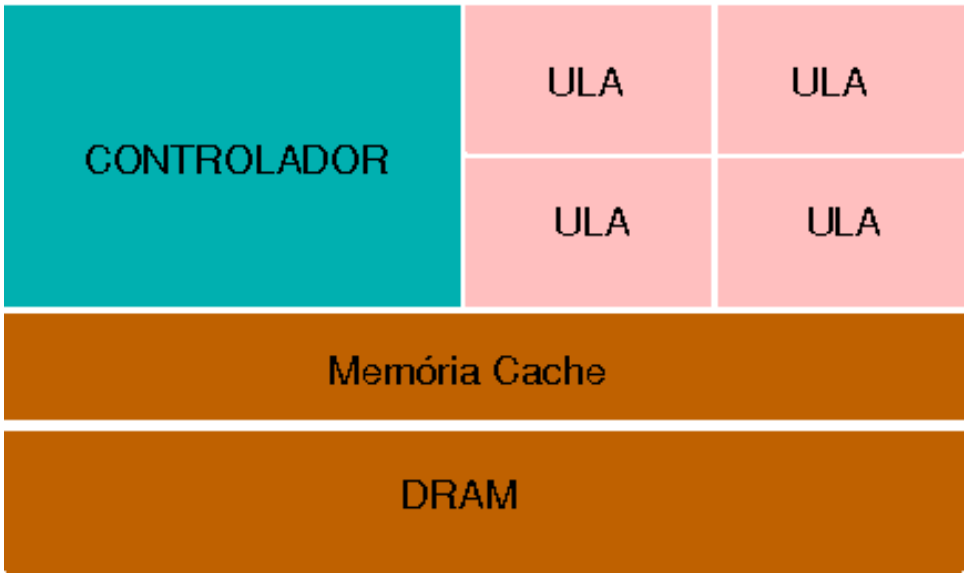


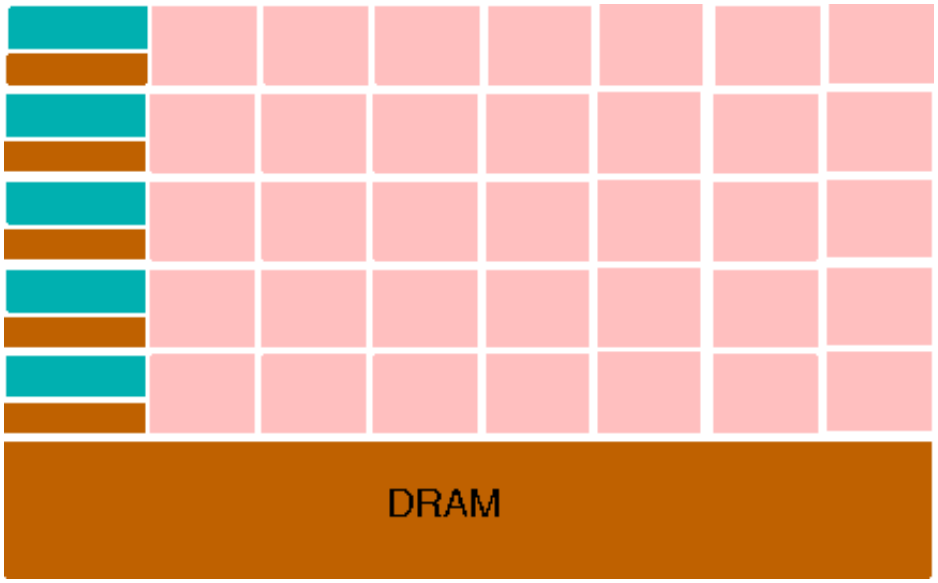
Fonte: http://www.tecmundo.com.br/hardware/1127-o-que-e-gpu-.htm

CPU x GPU

- CPU: arquitetura MISD ou MIMD
- Paralelismo em tarefas
- *Prefetch* de instruções

- GPU: arquitetura SIMD
- Paralelismo em dados
- *Prefetch* de dados



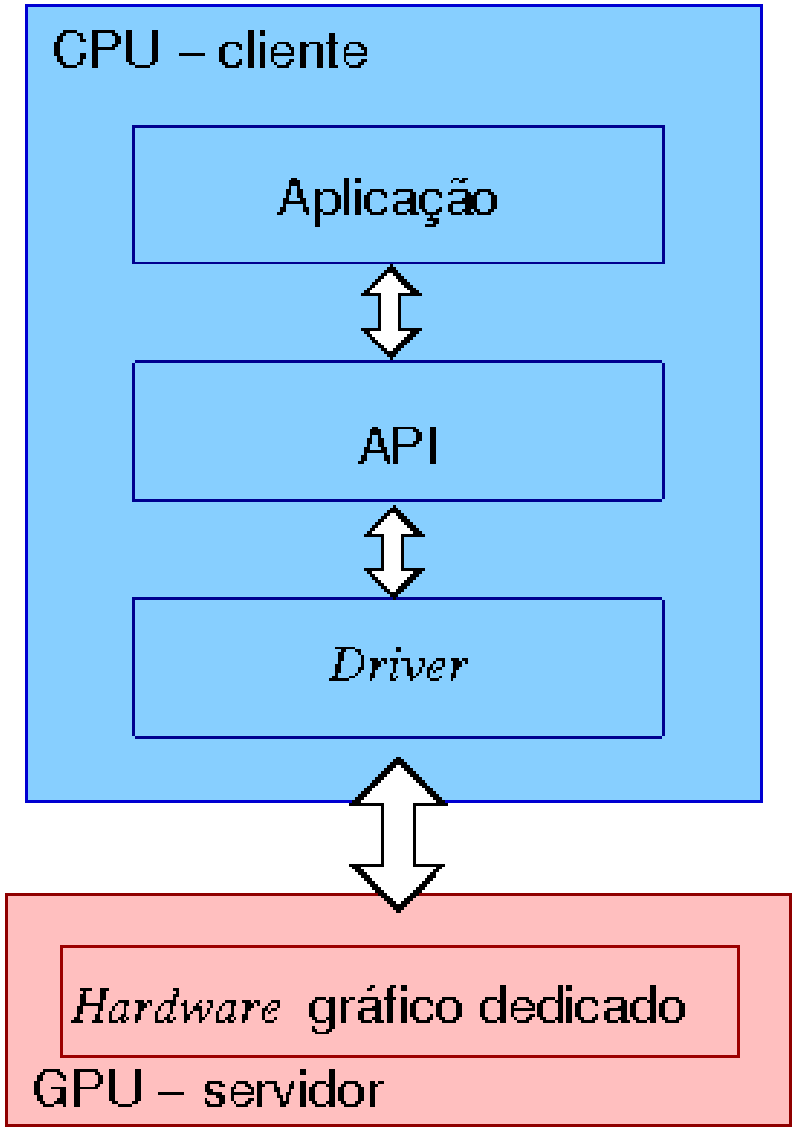


Fonte
http://selkie.macalester.edu/csinparallel/modules/GPUProgramming/build/html/Introduction/Introduction.html

CPU x GPU

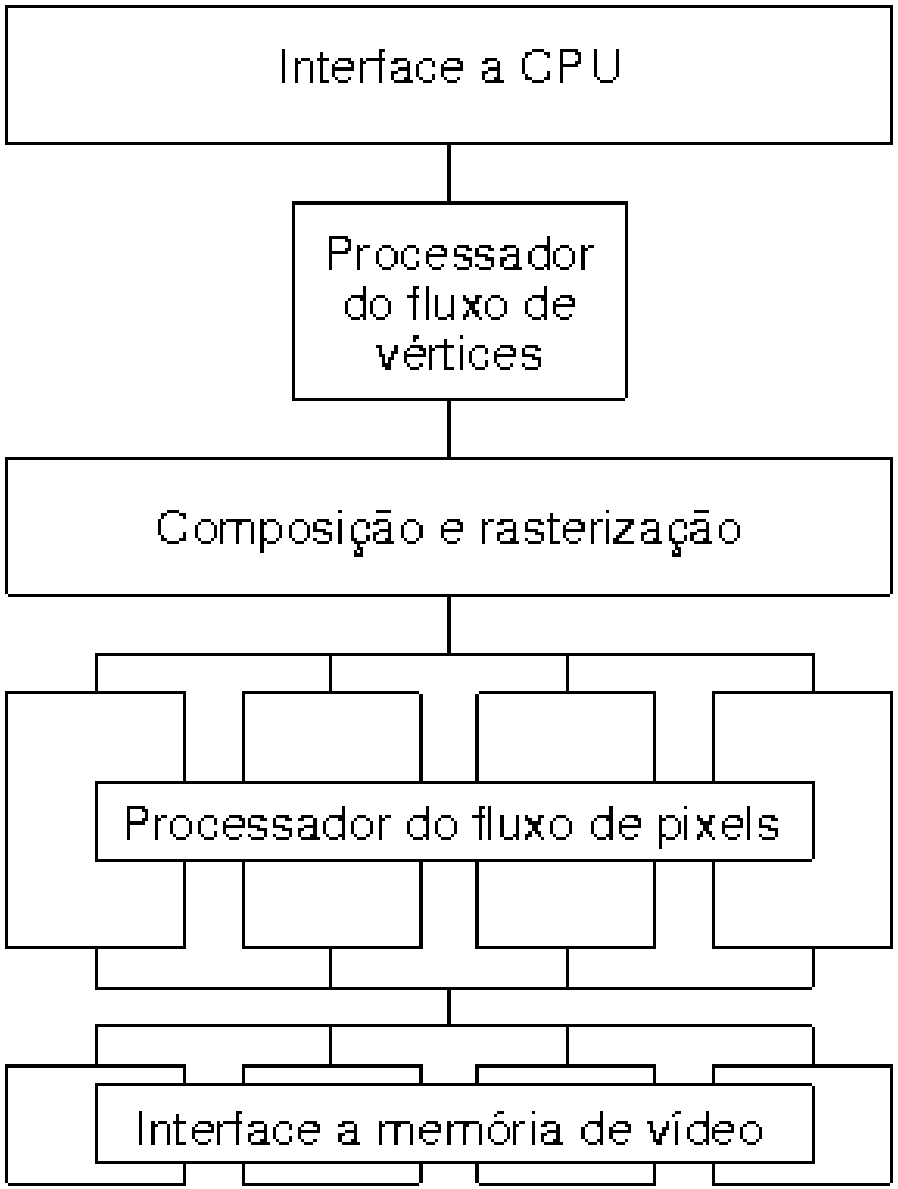
- http://www.flixxy.com/gpu-vs-cpu.htm

Cooperação entre CPU e GPU



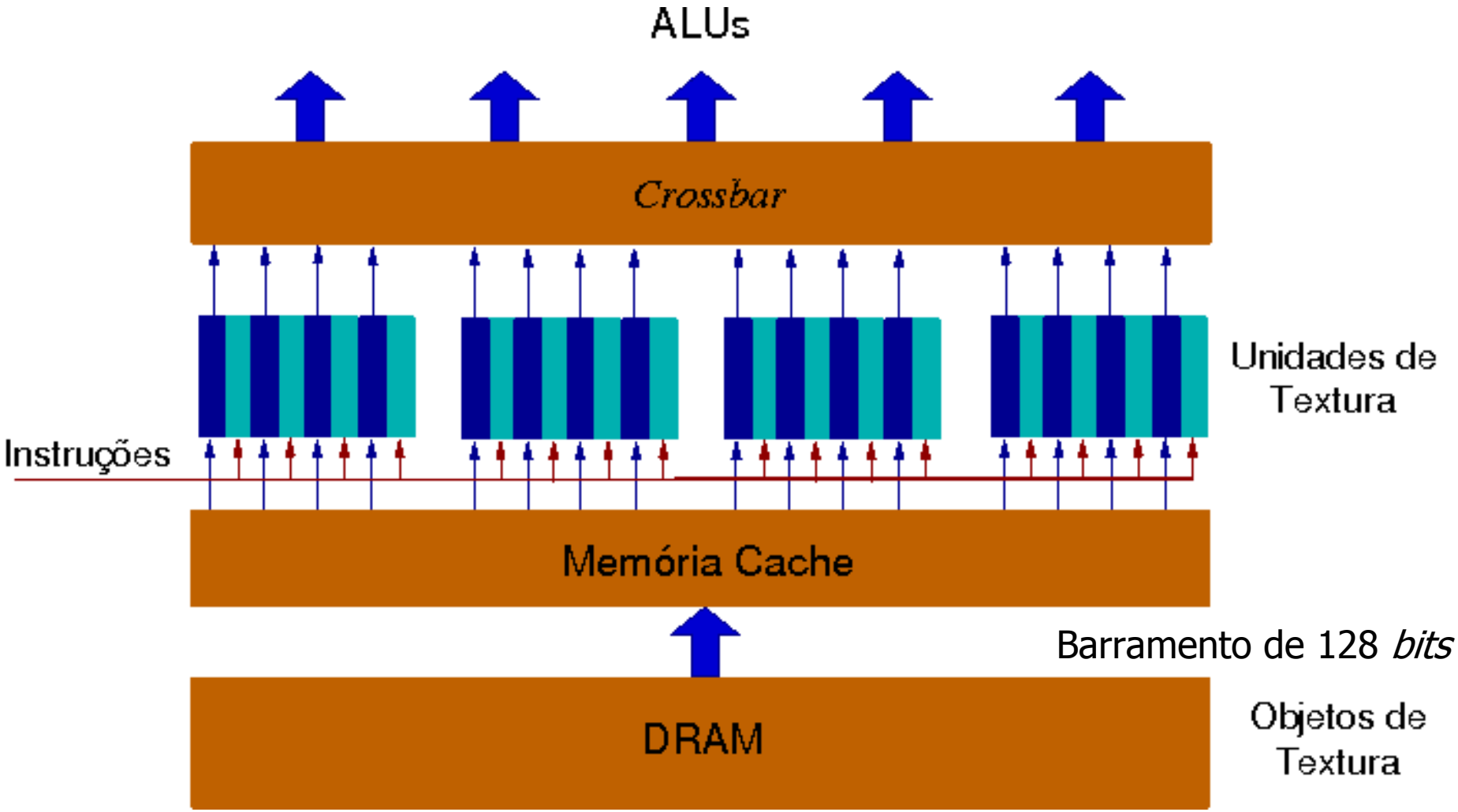
GPUs Programáveis

- Arquitetura GeForce 3
  - Processadores de vértice e de fragmento dedicados

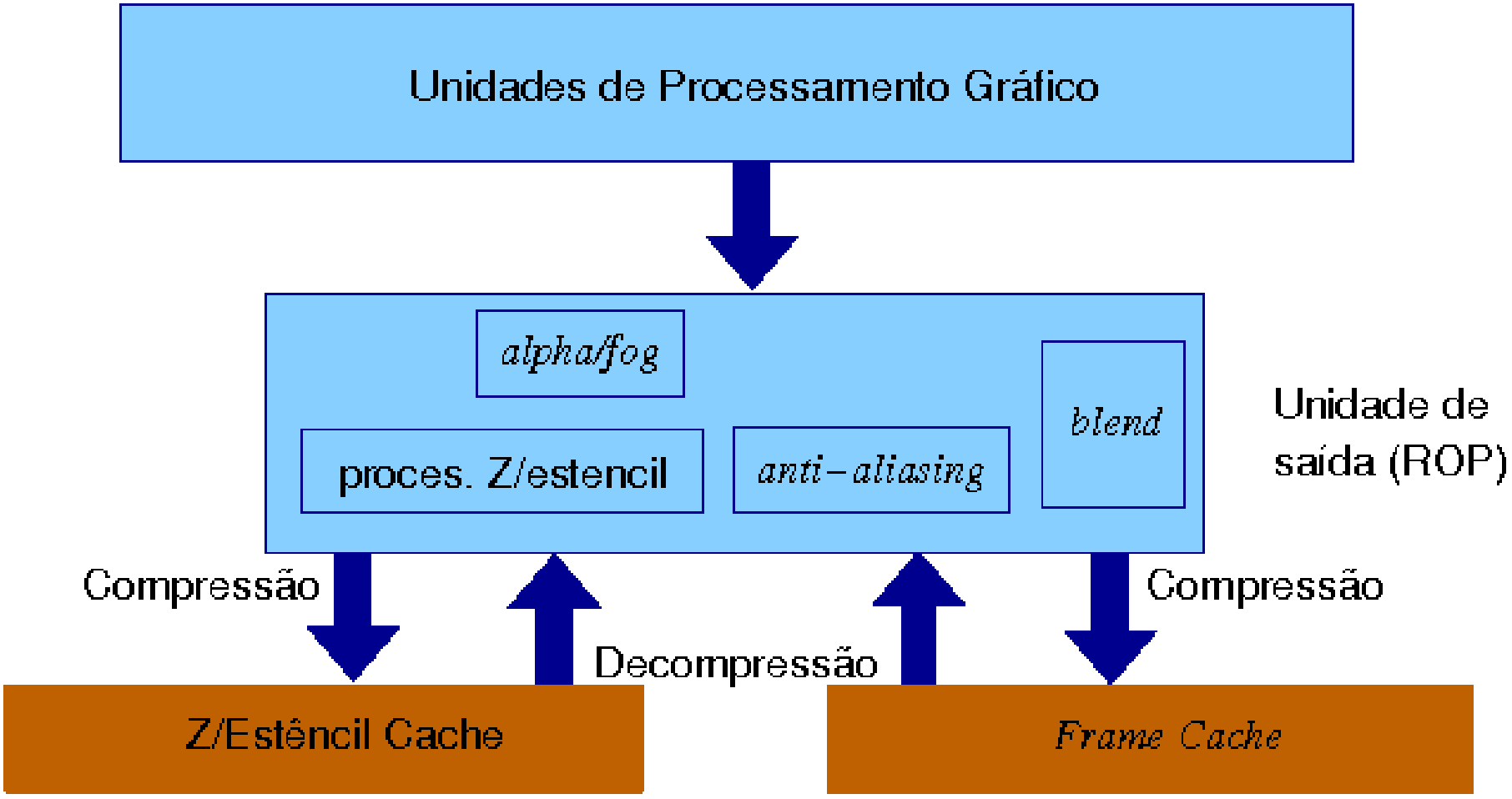


GPUs: Unidades de Memória de Textura

- Interpolações, Filtragem dos objetos de textura

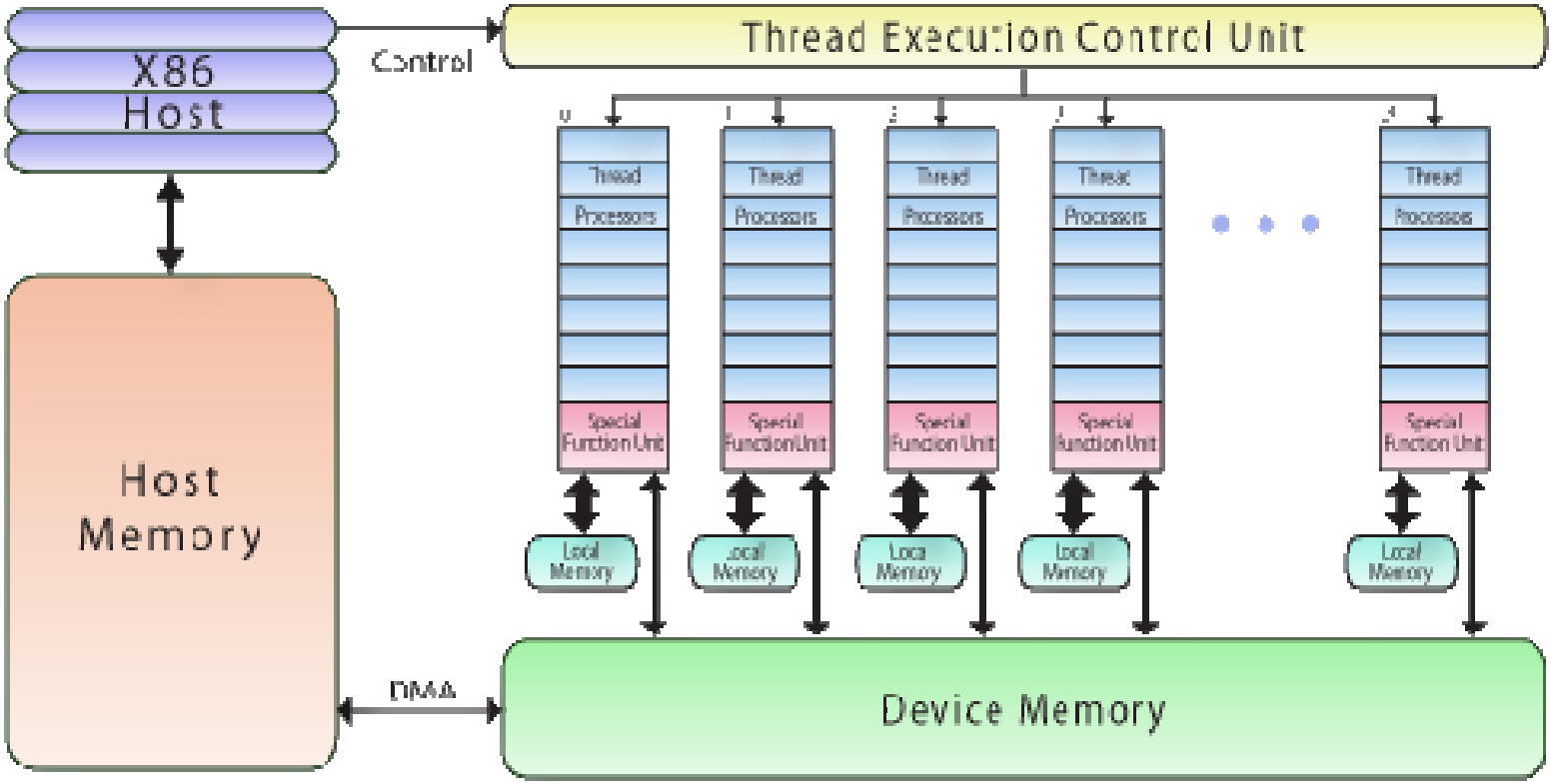


GPUs: Unidades de Saída (ROPs)



GPUs Programáveis Unificadas

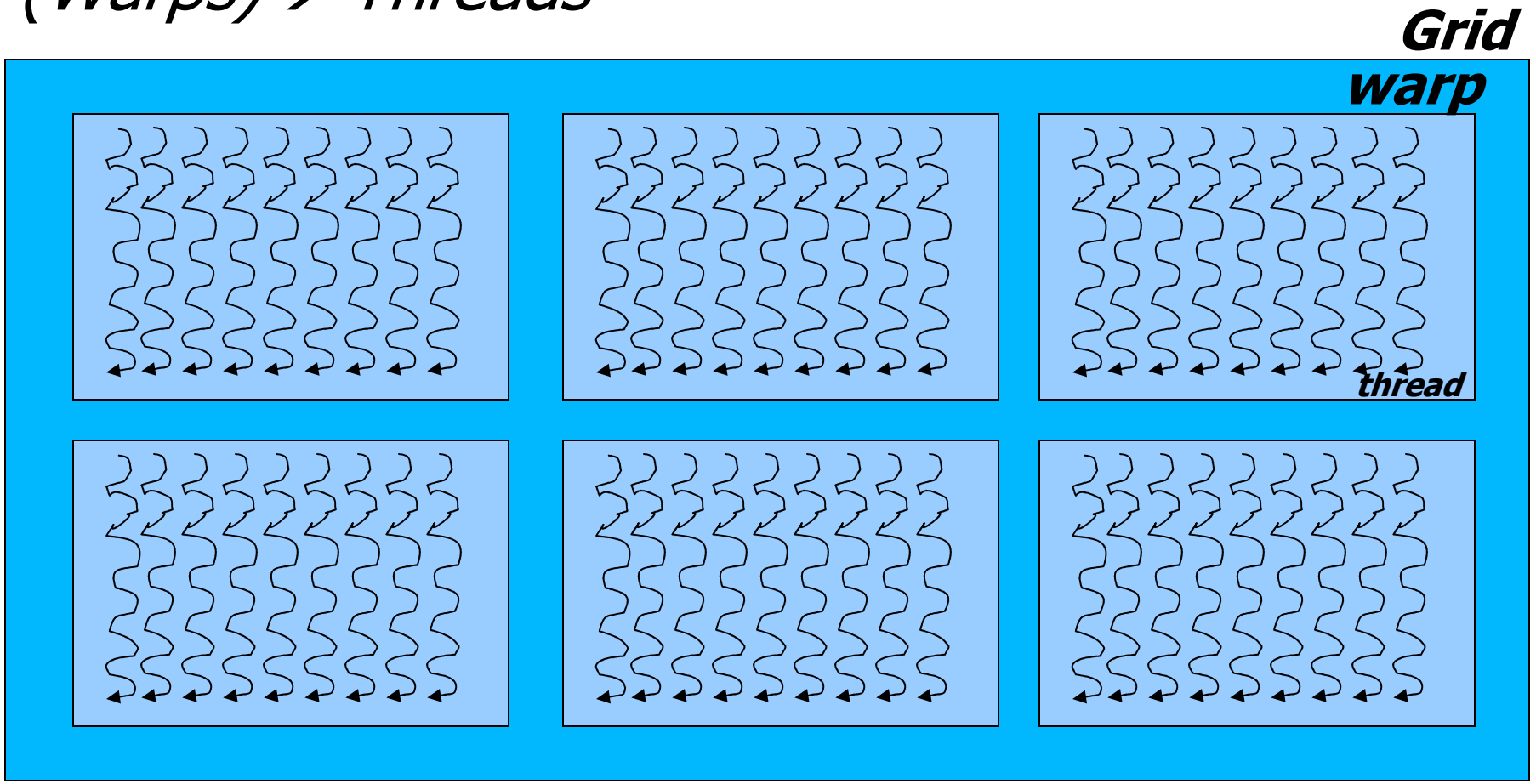
- Arquitetura Unificada:
  - GeForce 8 →Tesla e Fermi → Kepler



Fonte: http://www.pgroup.com/lit/articles/insider/v2n1a5.htm

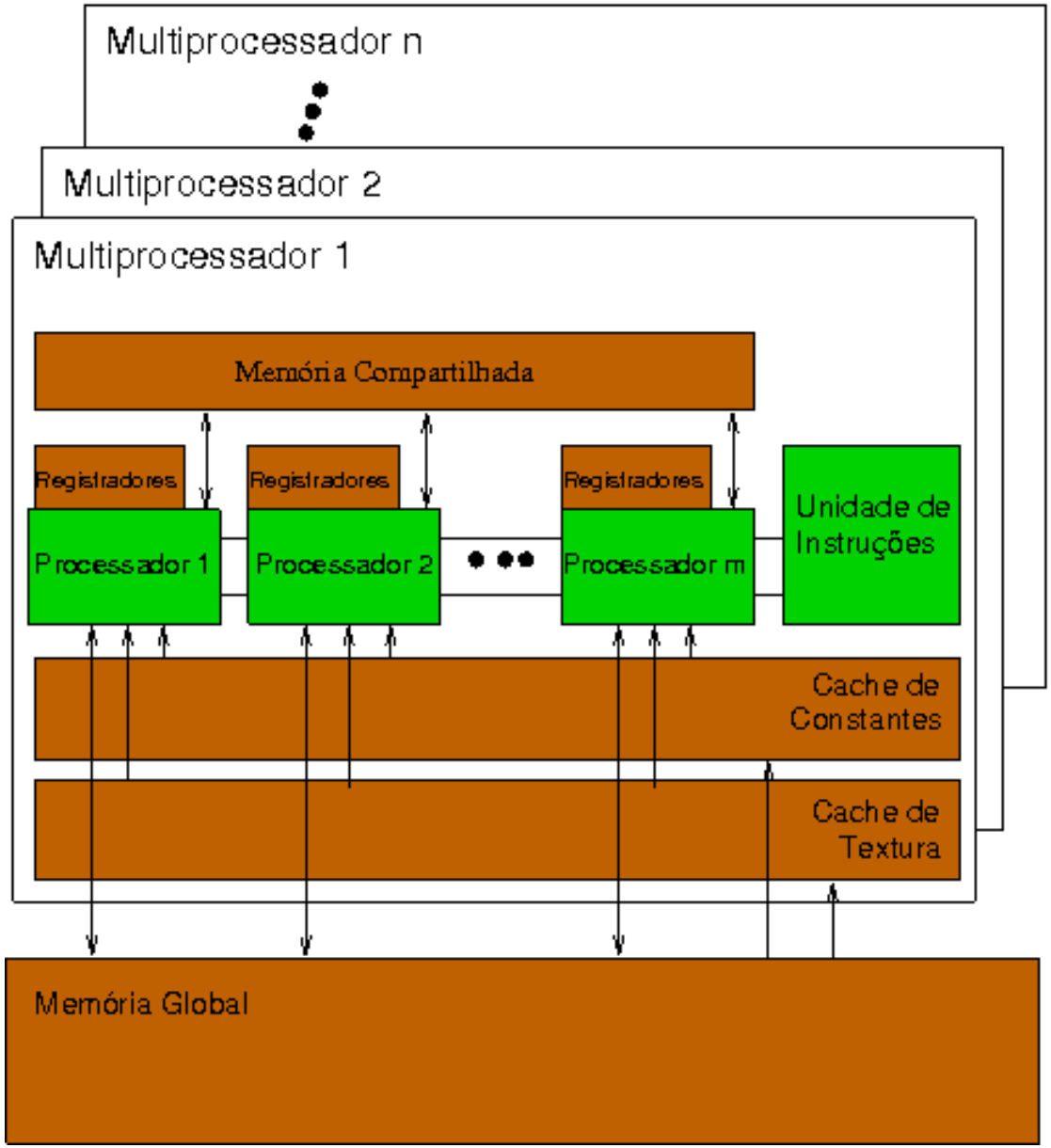
Fluxos de Execução

- *Streaming Multiprocessor (Grid) → SIMT processors (Warps)→ Threads*

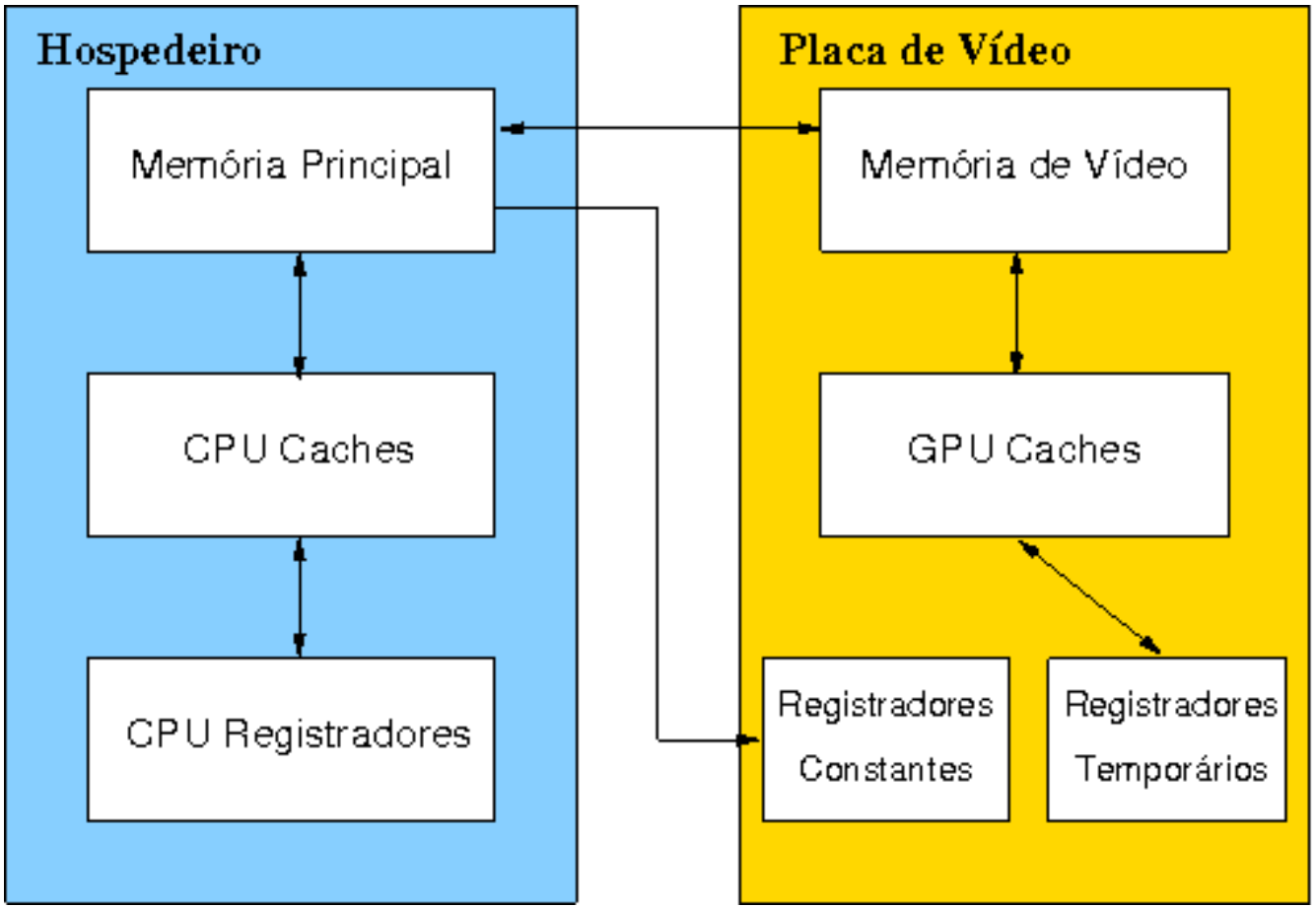


http://www.cs.cmu.edu/afs/cs/academic/class/15869-f11/www/readings/lindholm08_tesla.pdf

Hierarquia de Memória



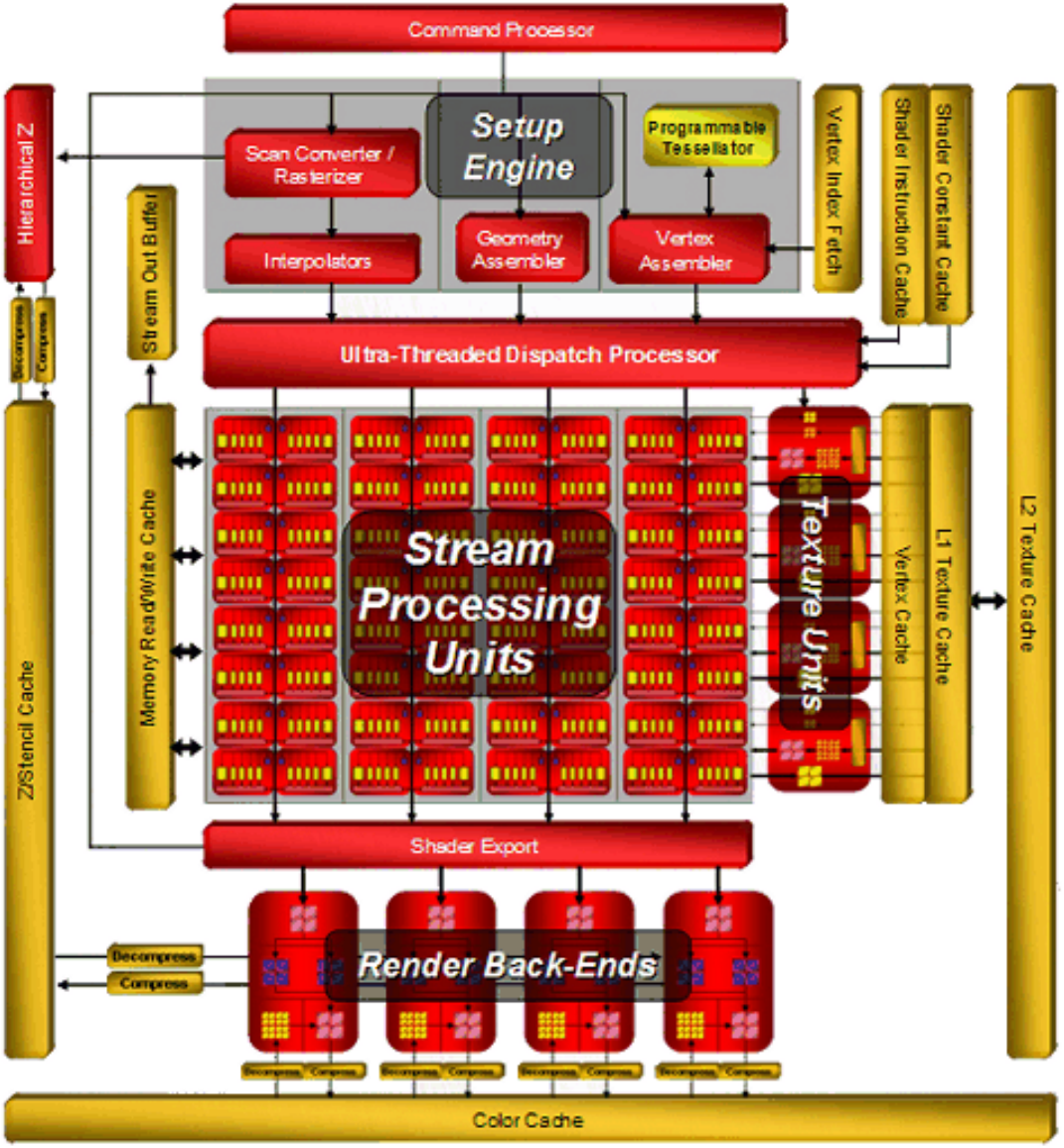
Espaços de Endereçamento



DISTINTOS!

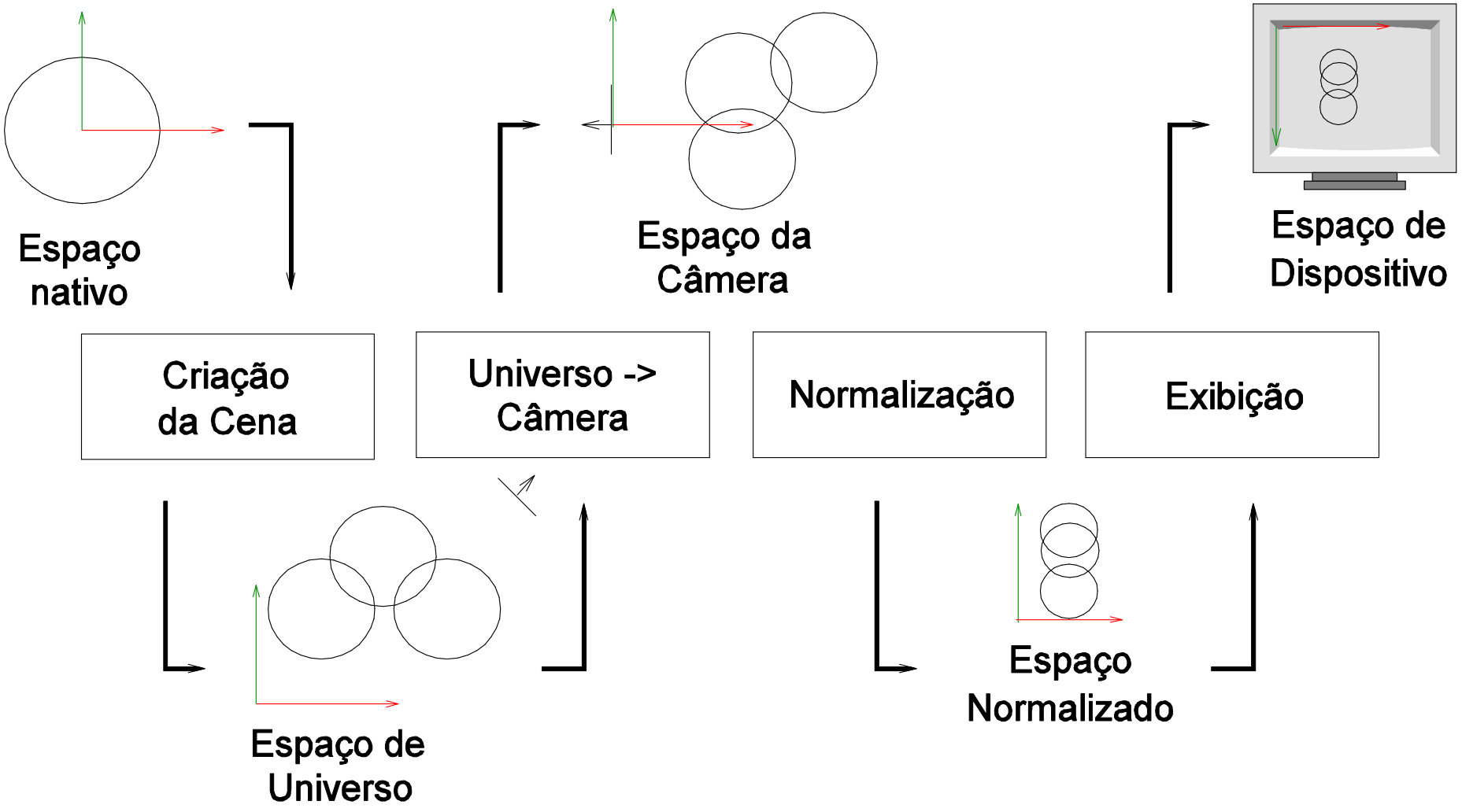
GPUs Programáveis: Síntese

- Radeon HD 2900XT



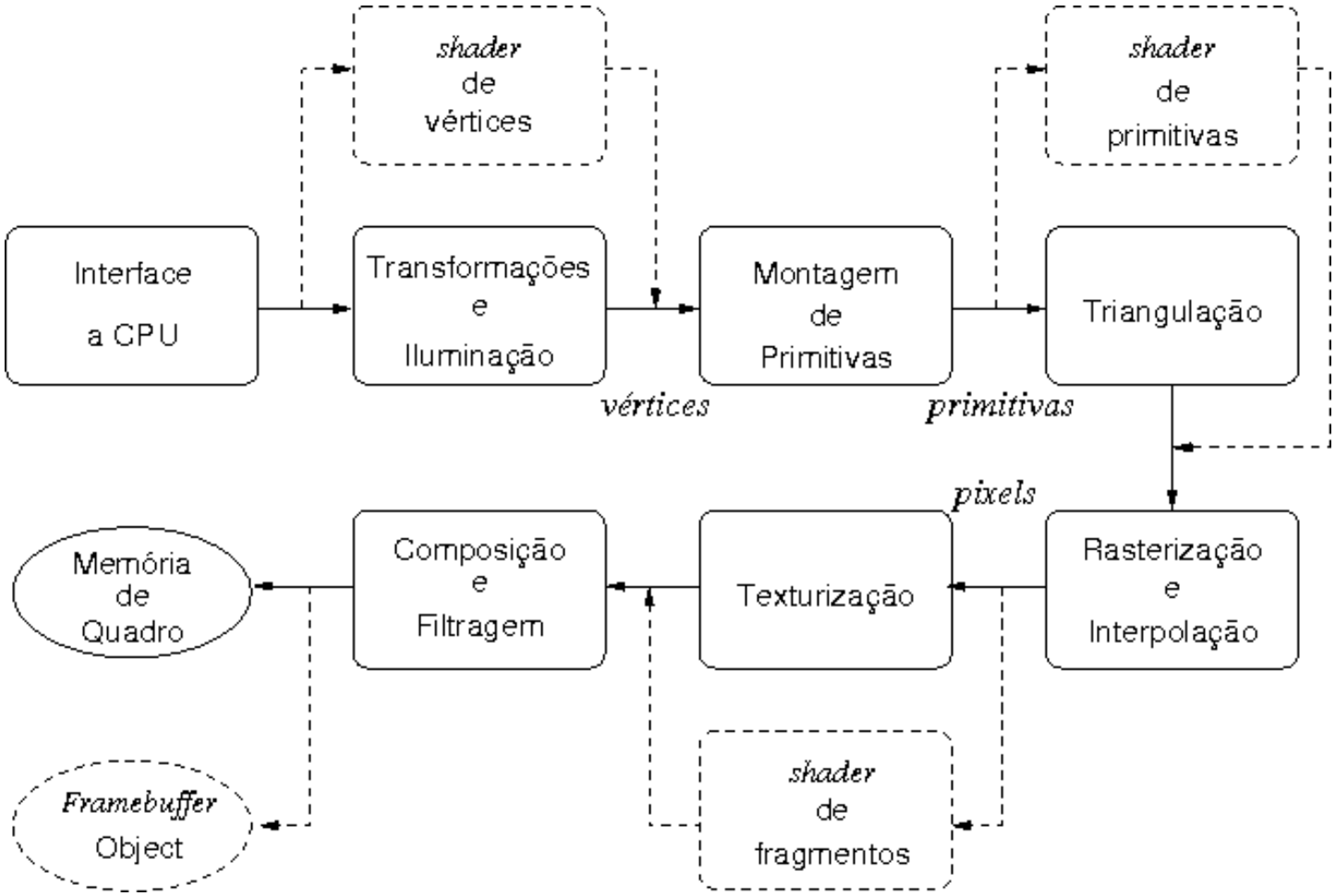
Fonte: http://ixbtlabs.com/articles2/video/r600-part1.html

... Relembrando: Fluxo de Renderização



OpenGL API

- Abstração em um fluxo de renderização



Diretrizes

1. Deve-se minimizar e simplificar a sequência de instruções em um *shader*.
2. Deve-se maximizar a localidade espacial dos dados na memória de textura.
3. Deve-se evitar estruturas de dados complexas.
4. Deve-se instruir apropriadamente as GPUs para otimizar o balanço enre os processamentos e os acessos às memórias.

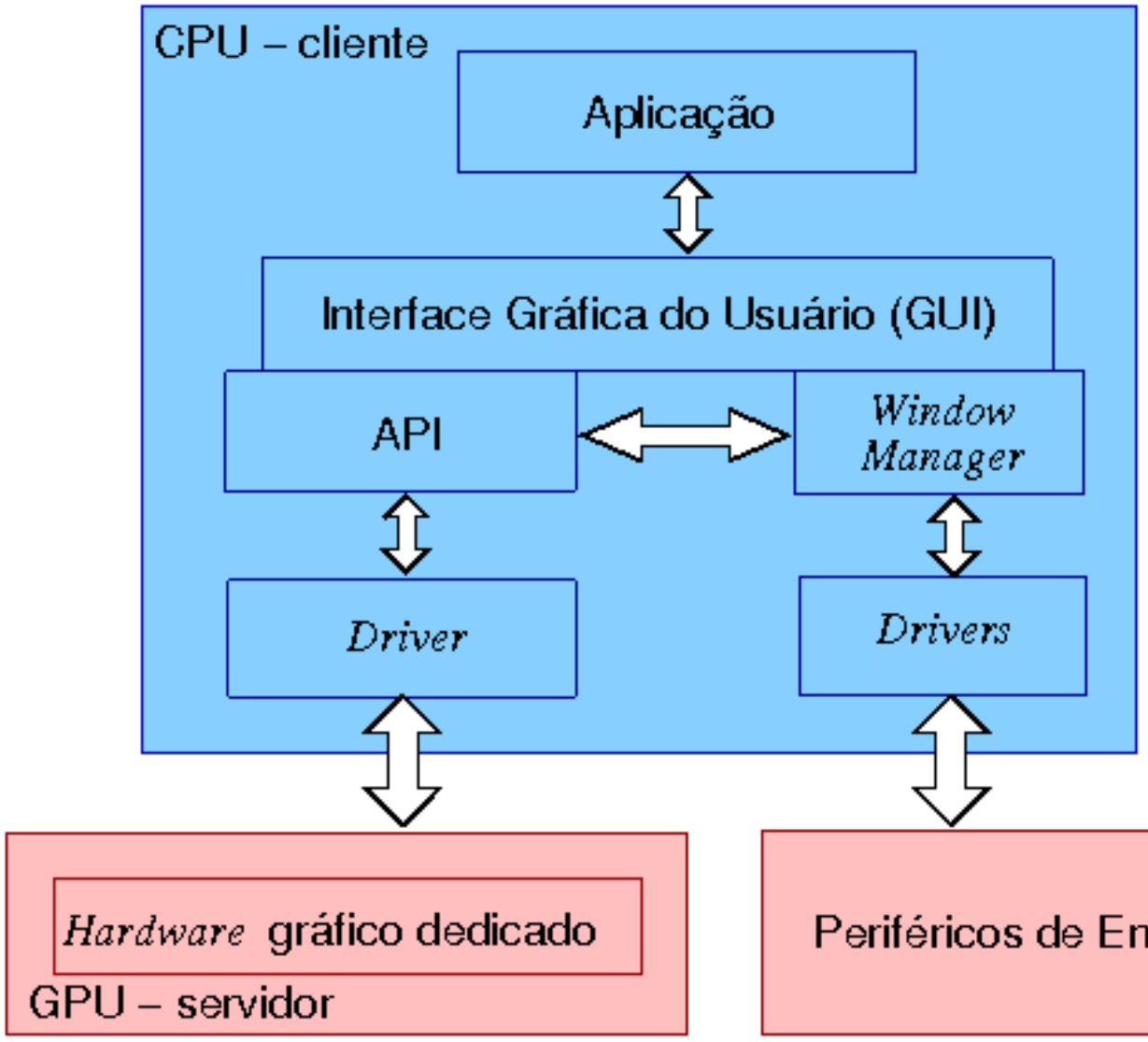
Linguagem de Programação Gráfica

- CPUs: OpenGL, Direct3D
  - Linguagem nativa: C
  - Interfaces com outras linguagens: Java, Python, Pascal, etc
- GPUs: *shading language* para renderização em tempo real
  - *ARB Assembly Language*
  - GLSL: *OpenGL Shader Language*
  - HSLS: *DirectX High-Level Shader Language*
  - Cg: *Cg Programming Language*

API OpenGL

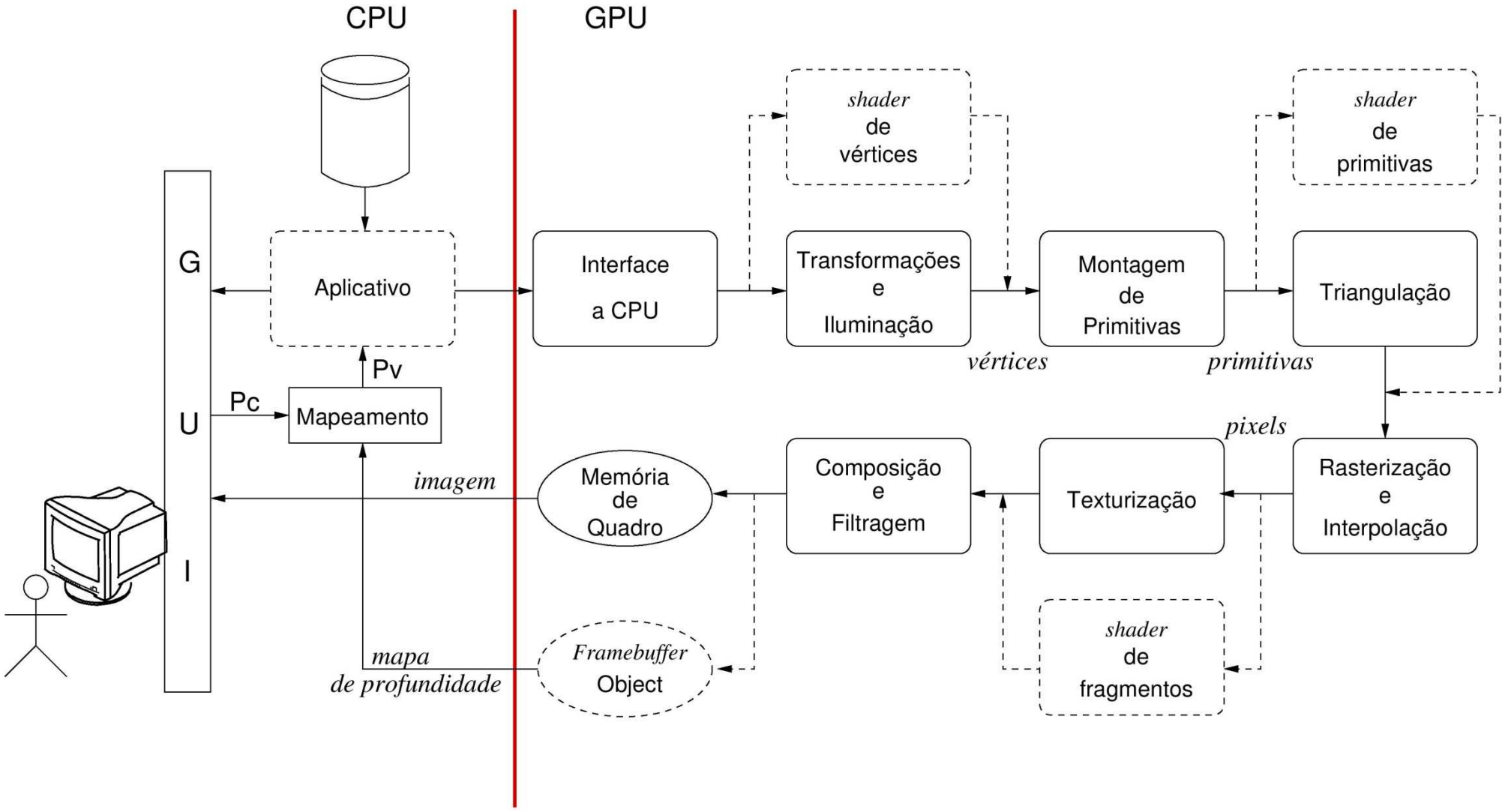
- Uma máquina de estados
- Tipos de dados similares a linguagem C
- Programabilidade
  - *Vertex shaders*: a partir de OpenGL 1.5/2.0, transformações geométricas, iluminação, coordenadas de textura
  - *Fragment shaders*: a partir de OpenGL 1.5/2.0, mapeamento de textura, *fog*
  - *Geometry shaders:* a partir de OpenGL 3.2
  - *Tesselation shaders*: a partir de OpenGL 4.
- Operação no modo imediato ou no modo retido
- Não gerencia recursos de janela

Interface Gráfica do Usuário (GUI)



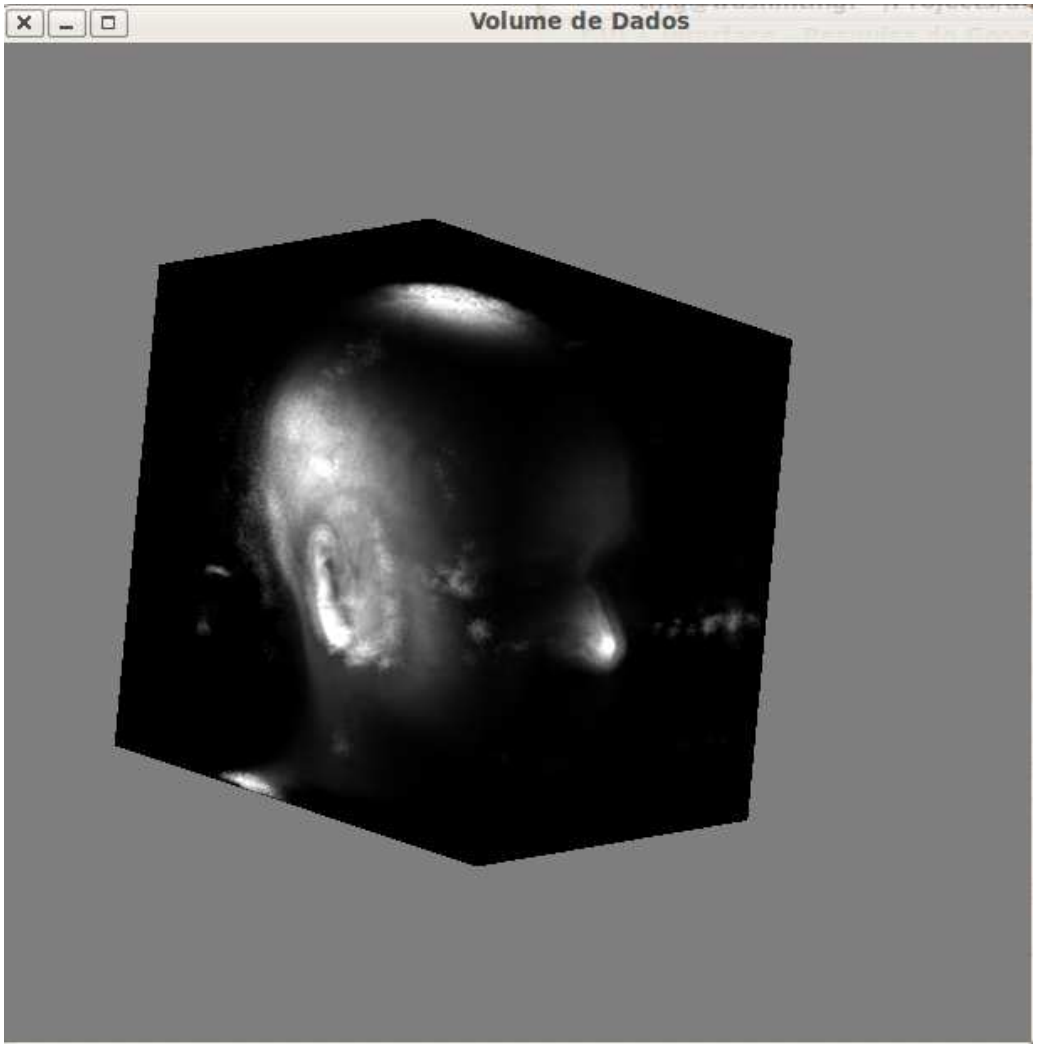
glut
FLTK
wxWidgets
Qt

Fluxo de Visualização



Ilustrando com um exemplo ...

- Renderizar um volume de dados no formato DICOM?



Requisitos de Aplicativos

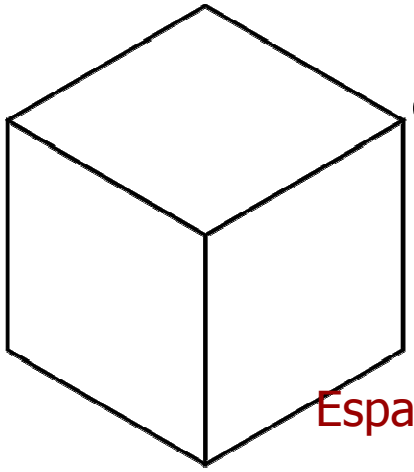
- Renderização: Versão de OpenGL >= 2.0
  - C++
    - std::cout << "OpenGL version supported by this platform: " << glGetString(GL_VERSION) << std::endl;
- Interface Gráfica do Usuário: GLUT
  - http://www.opengl.org/resources/libraries/glut/
  - https://users.cs.jmu.edu/bernstdh/web/common/help/cpp_mingw-glut-setup.php
  - http://sujatha-techie.blogspot.com.br/2008/10/glsl-with-mingw.html
- Importação de DICOM: GDCM
  - http://www.creatis.insa-lyon.fr/software/public/Gdcm/Main.html

... em Windows

- Compilador de C e C++ livre: MinGW
  - http://www.mingw.org
- Funcionalidades de OpenGL superior a 1.0: GLEW
  - http://glew.sourceforge.net/

Técnica baseada em textura

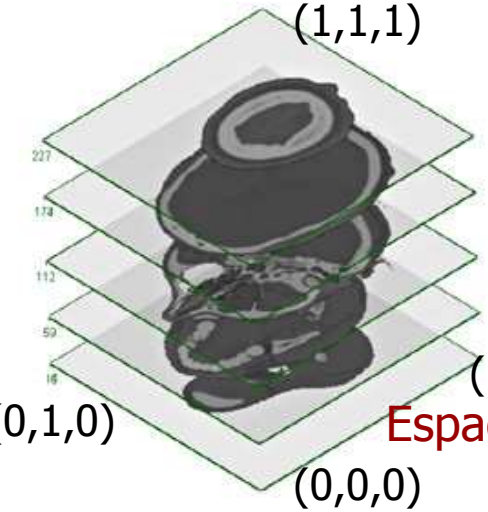
- "Texturizar" um cubo (*proxy geometry*) com o volume de dados de interesse



Proxy Geometry em dimensões reais do paciente

Espaço do objeto

(1,1,1)

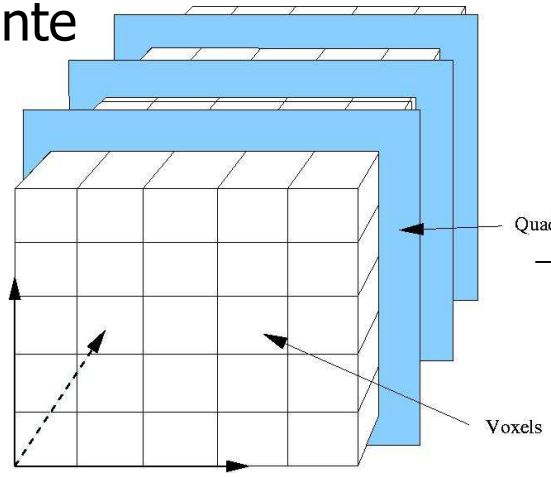


(1,0,0)

(0,1,0)

Espaço de textura

(0,0,0)



Alinhamento dos dois espaços pelas coordenadas de textura e coordenadas espaciais dos vértices

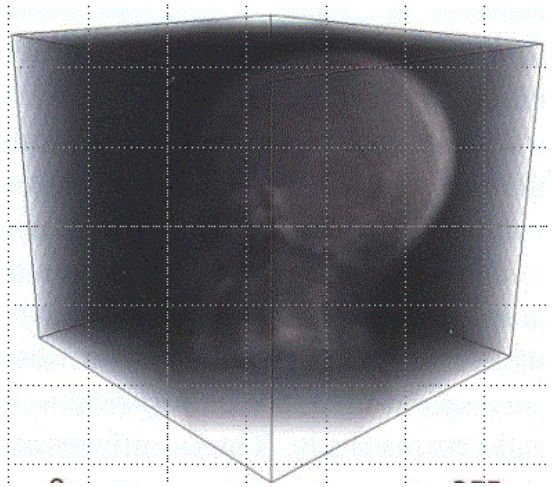


Imagem Final

Solução 1: OpenGL 1.2

- "Normalize" os valores das amostras:
  - data.buffer$\in$[mínimo,máximo] → texbuffer$\in$[0,255]
- Carregue o volume de dados na memória de textura

```
glBindTexture(GL_TEXTURE_3D, texID[0]);
glTexImage3D(GL_TEXTURE_3D, 0, GL_LUMINANCE, data.dims[0],
             data.dims[1], data.dims[2], 0, GL_LUMINANCE,
             GL_UNSIGNED_BYTE, texbuffer);
```

- Associe a cada vértice coordenadas de textura

```
glTexCoord3f(0.0, 0.0, 0.0);
glVertex3f(0.0, 0.0, 0.0);
```

Interfaceamento com WM

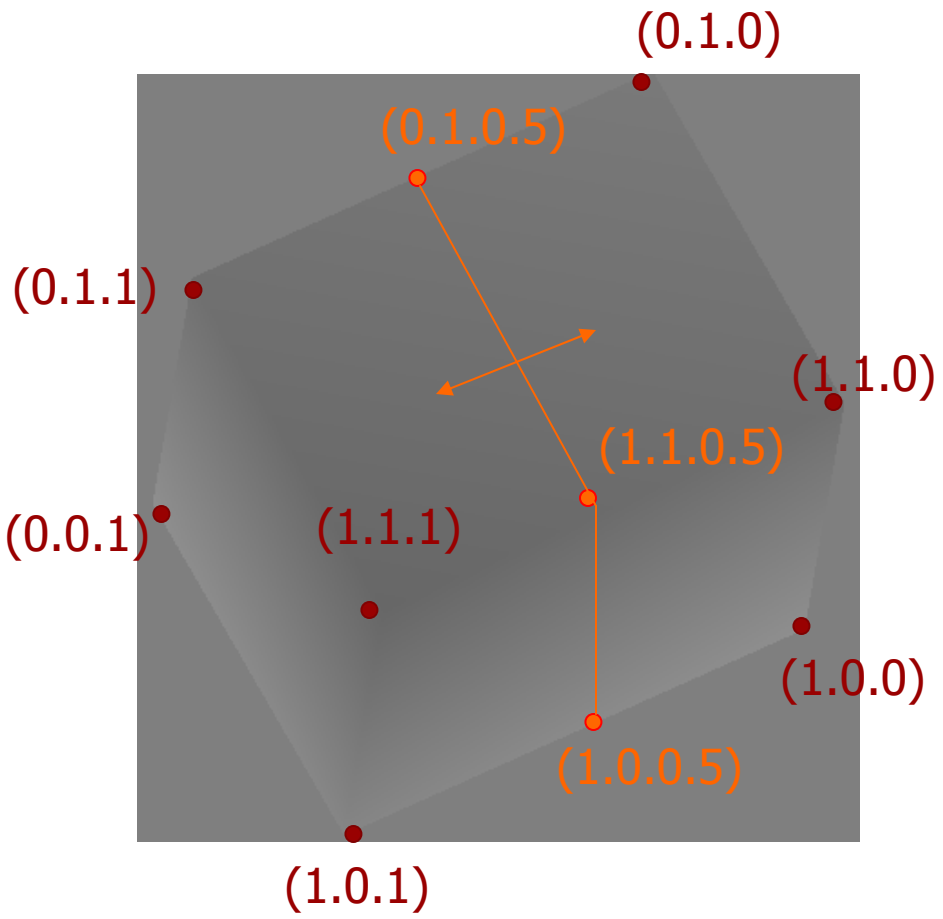
- Funções de glut

```
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_SINGLE);
win = glutCreateWindow("Volume de Dados");
init();
glutPositionWindow(10, 10);
glutReshapeWindow(VIEWPORT_WIDTH, VIEWPORT_HEIGHT);
glutReshapeFunc(reshape);
glutDisplayFunc(desenheProxyGeometry);
glutKeyboardFunc(keyboard);
glutMainLoop();
```

Interações com *Keyboard*

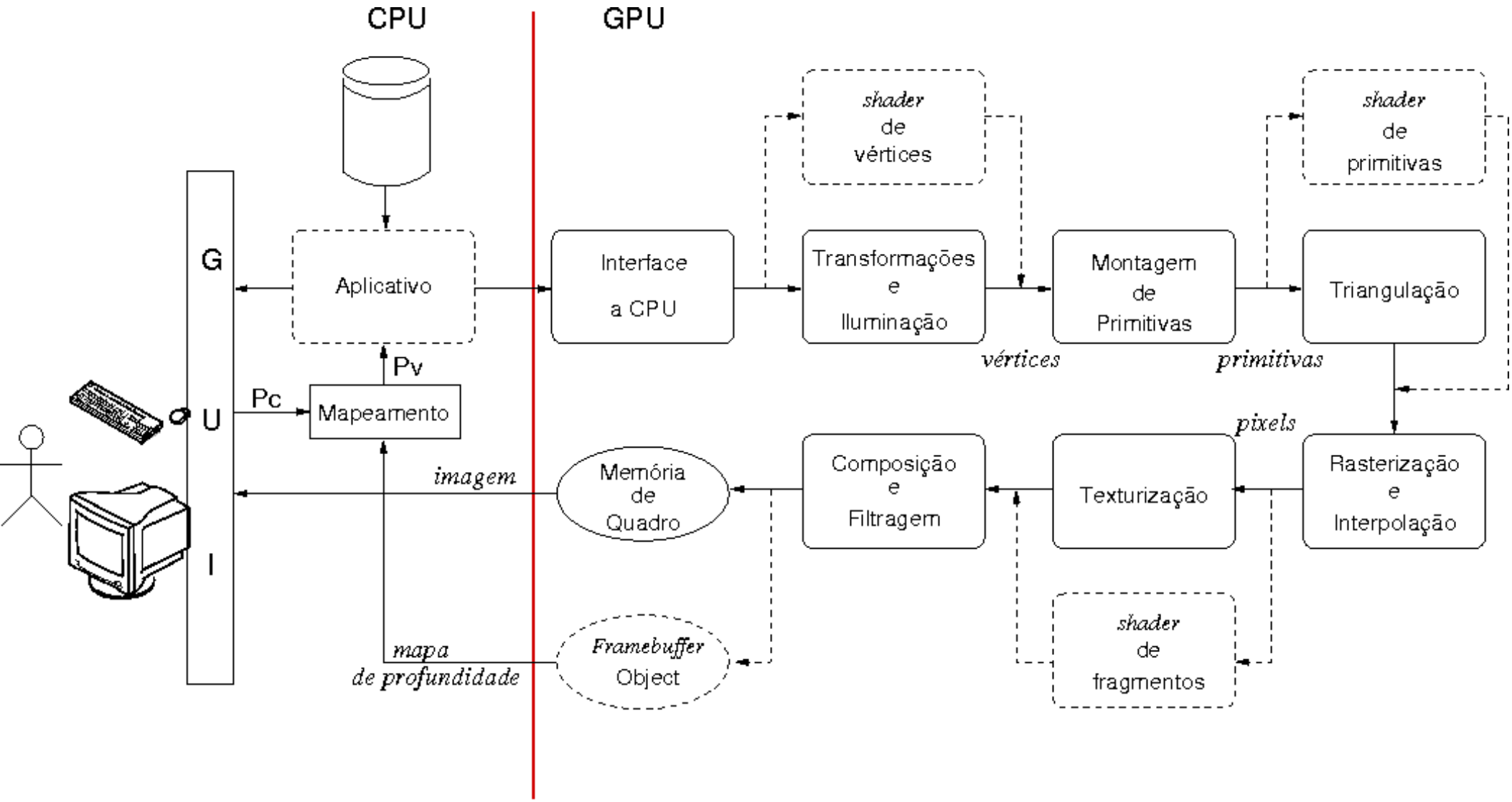
```
void keyboard(unsigned char key, int x, int y) {
switch (key) {
:
case '+':
      if (nVolSideZ + 0.05 < 1.0) nVolSideZ += 0.05;
      else nVolSideZ = 1.0;
      break;
case '-':
      if (nVolSideZ - 0.05 > 0.0) nVolSideZ -= 0.05;
      else nVolSideZ = 0.0; break;
case 27:
      :
      exit(0);
      return;
} }
```

Estrutura Interna




```
glTexCoord3f(0.0, 1.0, 0.0);
glVertex3f(0.0, nVolSides[1], 0.0);
:
glTexCoord3f(0.0, 1.0, nVolSideZ);
glVertex3f(0.0, nVolSides[1],
    nVolSideZ*nVolSides[2]);
:
glTexCoord3f(1.0, 1.0, nVolSideZ);
    glVertex3f(nVolSides[0],
    nVolSides[1],
    nVolSideZ*nVolSides[2]);
```

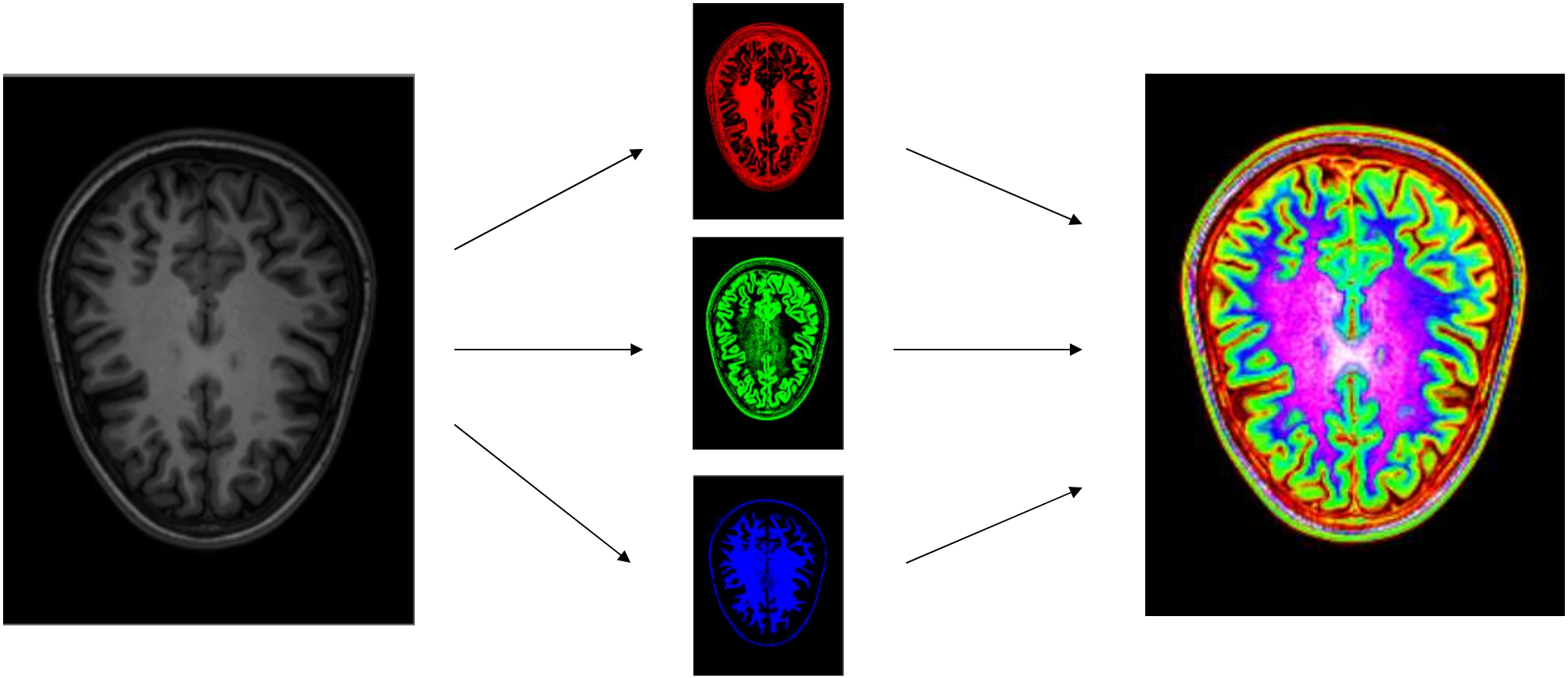
Fluxo de Visualização Interativa



Demo jai13_basic

Solução 2: Imagens Coloridas

- Como alterar o mapeamento dos valores das amostras para diferentes cores?
  - Ao invés de um valor escalar, associe um vetor RGB



Solução 2: OpenGL 1.2

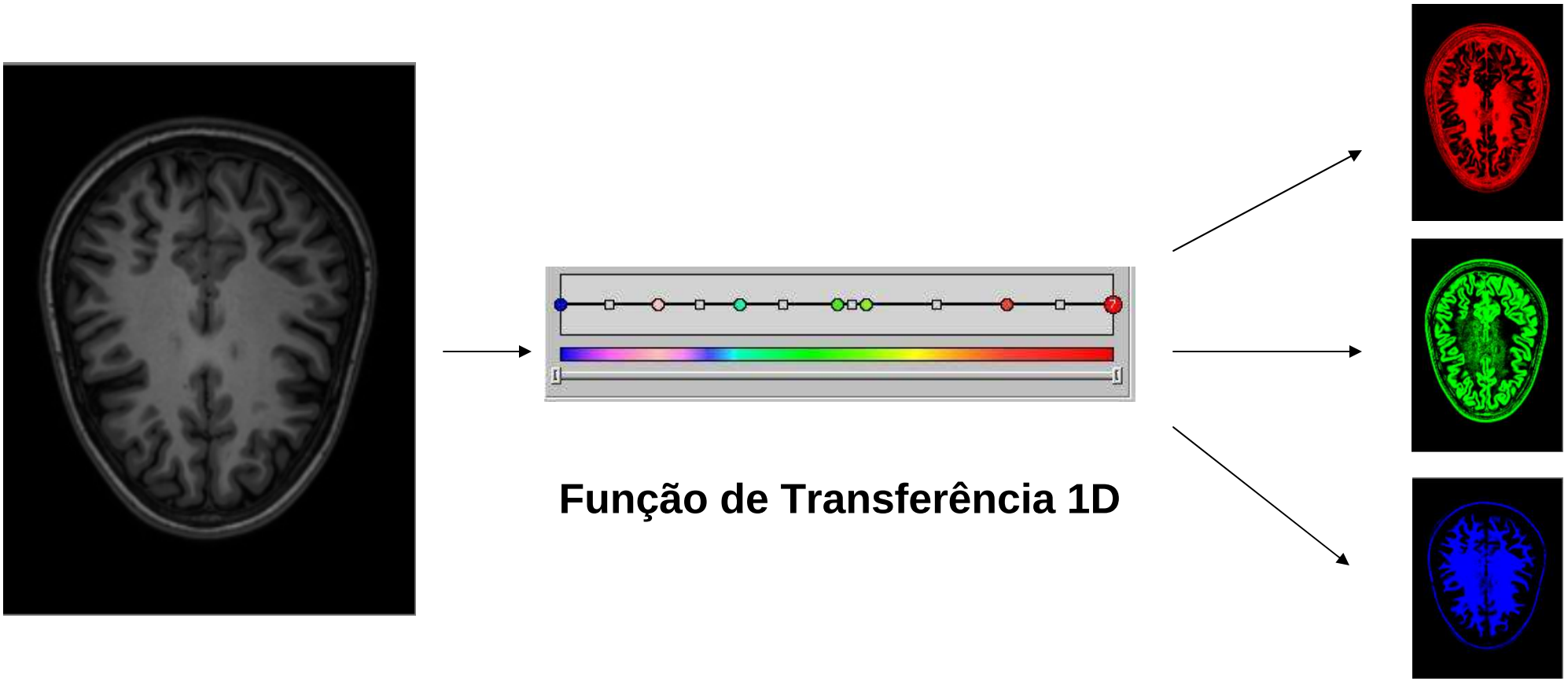
```
if (intensidade < 7) {
        texbuffer[di] = texbuffer[di+1] = texbuffer[di+2] = 0;
} else if (intensidade < 14) {
        texbuffer[di] = 51; texbuffer[di+1] = texbuffer[di+2] = 0;
} else if (intensidade < 21) {
        texbuffer[di] = 102; texbuffer[di+1] = texbuffer[di+2] = 0;
:
:
}

glTexImage3D(GL_TEXTURE_3D, 0, GL_RGB, data.dims[0],
            data.dims[1], data.dims[2], 0, GL_RGB,
            GL_UNSIGNED_BYTE, texbuffer);
```

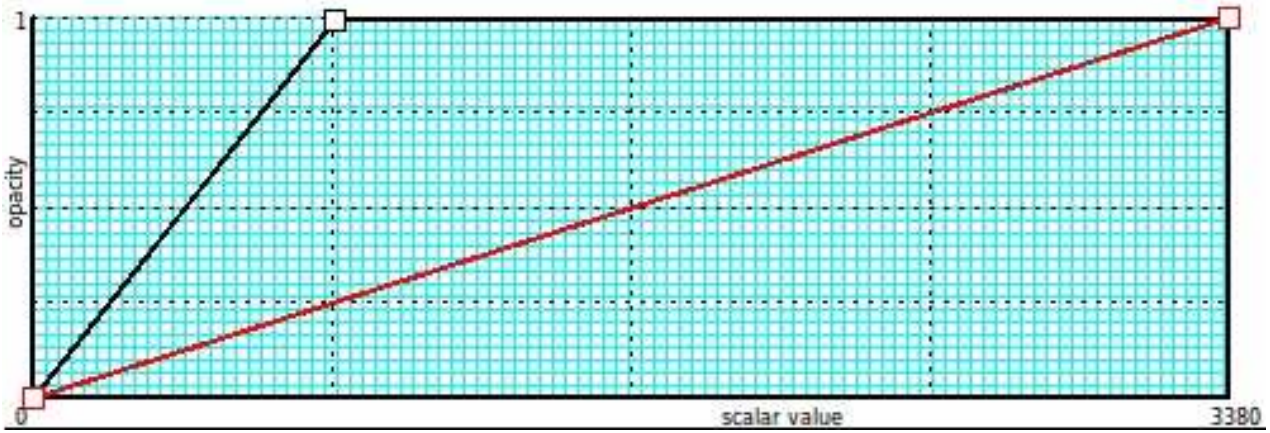
Demo jai13_basic_cor

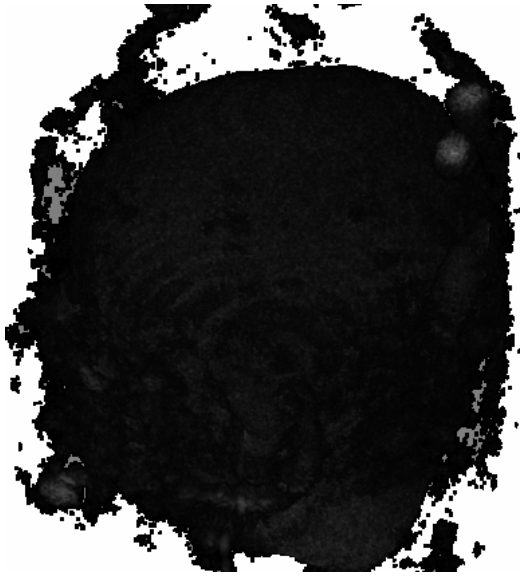
Solução 3: Função de Transferência

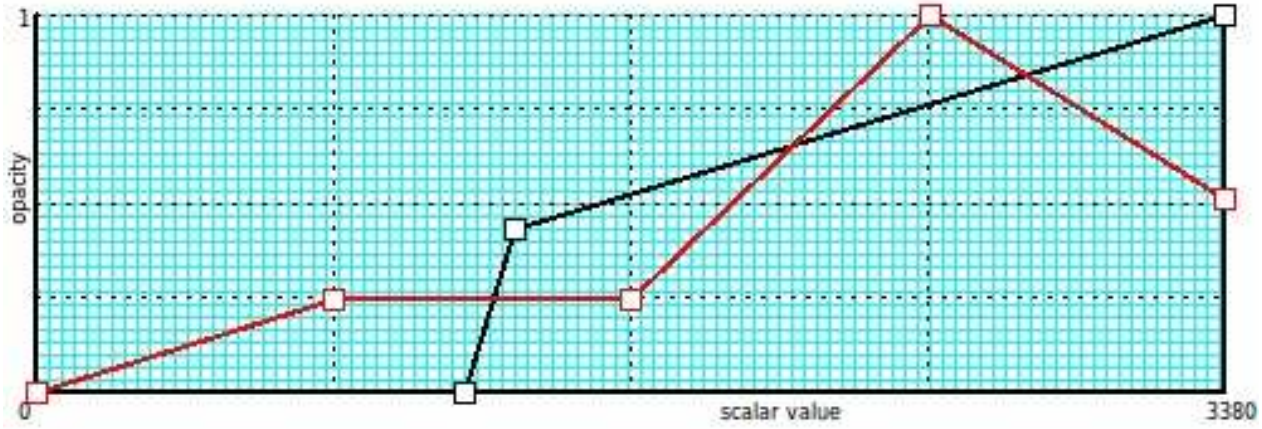
- Como tornar mais modulável o mapeamento?



Funções de Transferência







Solução 3: OpenGL 1.2

```
tfunc.GetGrayScaleTF (1, data.umin, data.umax, &dim, &tf);


intensidade = (int)((reinterpret_cast<unsigned
    short*>(data.buffer))[iz*data.dims[0]*data.dims[1]+iy*data.dims[0]+ix
    ]);
texbuffer[di++] = (unsigned char)(tf[intensidade-data.umin+1]*2);


glTexImage3D(GL_TEXTURE_3D, 0, GL_LUMINANCE, data.dims[0],
             data.dims[1], data.dims[2], 0, GL_LUMINANCE,
             GL_UNSIGNED_BYTE, texbuffer);
```

Demo jai13_basic_ft

Problema

- Quantidade de instruções a serem transferidas para a GPU no momento de renderização cresce com a quantidade de primitivas gráficas.

Solução 4: OpenGL 1.2

- Compacte um conjunto dos dados de vértices em arranjos em CPU

```
glEnableClientState (GL_VERTEX_ARRAY);

glVertexPointer (3, GL_FLOAT, 0, vertices);  // coord. compactadamente armazenadas

glEnableClientState (GL_TEXTURE_COORD_ARRAY);

glTexCoordPointer (3, GL_FLOAT, 0, texCoords0); // coord. compactamente armazenadas
```

- Desreferenciá-los no momento de transferência para GPU (renderização)

```
glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE,
   quadIndices);
```

Demo jai13_varray

Problema

- Quantidade de dados a serem transferidos para a GPU no momento de renderização continua grande.

Solução 5: OpenGL 1.4

- Aloque dados em GPUs: *vertex buffer objects*

```
glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
glBufferData(GL_ARRAY_BUFFER, sizeof(texCoords0), texCoords0,
    GL_STATIC_DRAW);
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vbo[2]);
glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(quadIndices), quadIndices,
    GL_STATIC_DRAW);
```

- Especifique o local dos dados em GPUs

```
glVertexPointer (3, GL_FLOAT, 0, (const GLvoid *)0);
glTexCoordPointer (3, GL_FLOAT, 0, (const GLvoid *)0);
```

- Renderize

```
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, vbo[2]);
glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, (const GLvoid *)0);
```

Demo jai13_buffer

OpenGL 1.4: Problema

- Estruturas internas só podem ser visualizadas em em cortes planares.

Organização do Minicurso

- Protocolo de diagnóstico de displasia cortical focal
- Modelo matemático de neuroimagens e técnicas de renderização
- Tecnologia de renderização: GPUs
- Estado-de-arte de visualização de neuromagens
- Implementação de ferramentas de interação