Pandora's Box Graphics Engine

Uma engine gráfica com aplicações em visualização científica

Andrew T. N. Kurauchi

Victor K. Harada

Orientador: Prof. Dr. Marcel Parolin Jackowski

Objetivos

- Construir uma engine de fácil aprendizado e utilização
- Aplicar técnicas avançadas de computação gráfica
- Desenvolver um visualizador da representação elipsoidal de campos tensoriais

OpenGL - O que é?

- Uma especificação aberta de interface de software para o hardware gráfico (GPU)
- Máquina de estados

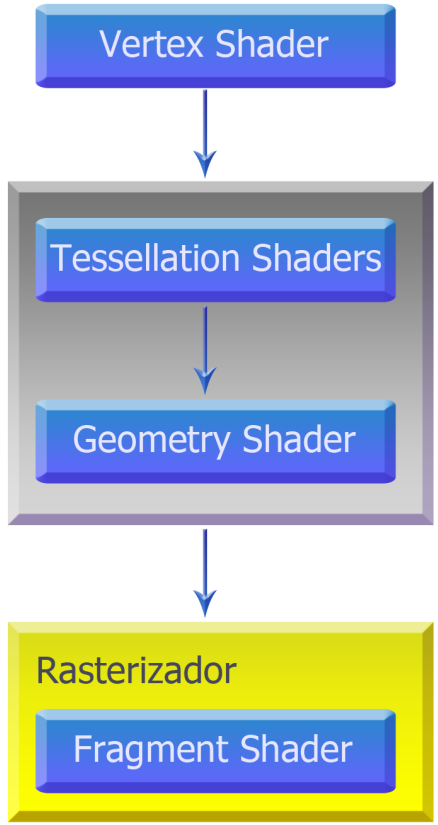
OpenGL - O que ele não faz

- Gerenciamento de janelas
- Tratamento de arquivos

OpenGL - O que ele faz

- Criação de formas a partir de primitivas (pontos, retas e polígonos)
- Mapeamento de primitivas (3D) para um buffer (2D)

Pipeline

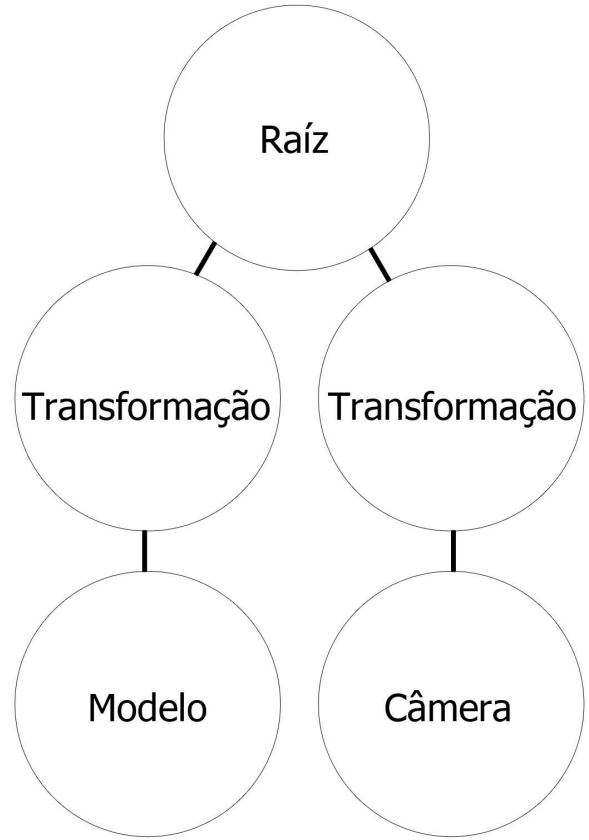

Vertex Shader
Tessellation Shaders
Geometry Shader
Rasterizador
Fragment Shader

Wrappers para os objetos do OpenGL

- Buffer
- Texture1D
- Texture2D
- TextureBuffer
- VertexBuffer
- Shader
- Program

Grafo de cena

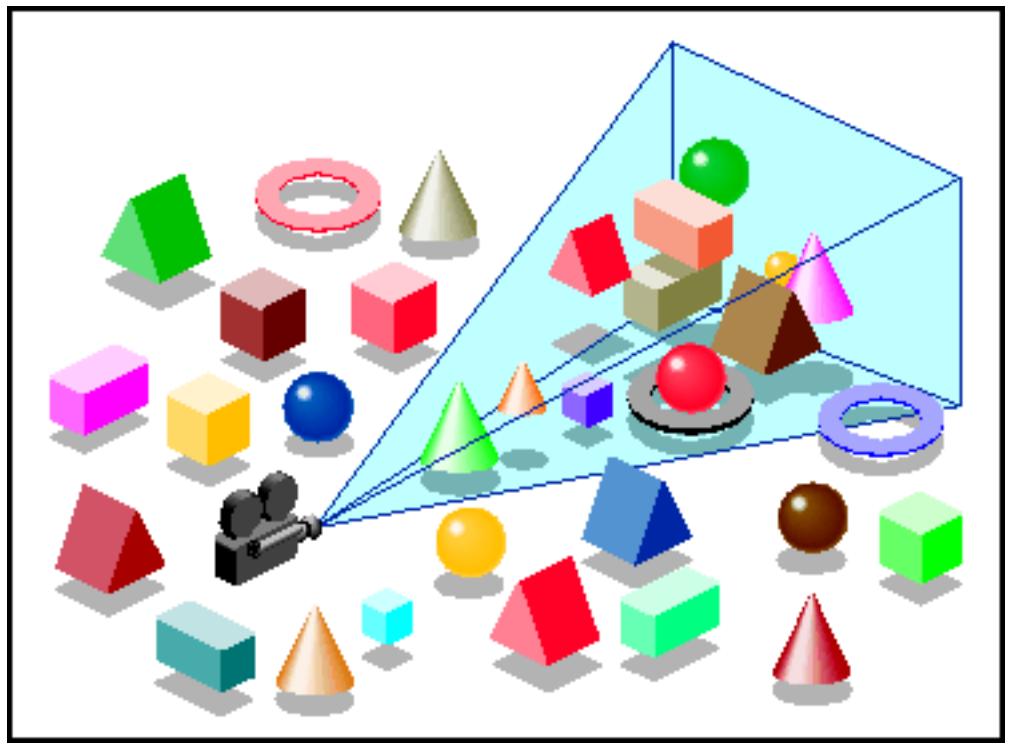
Para poder lidar com estruturas de cenas complexas a Pandora's Box utiliza um grafo de cena.



Renderizador

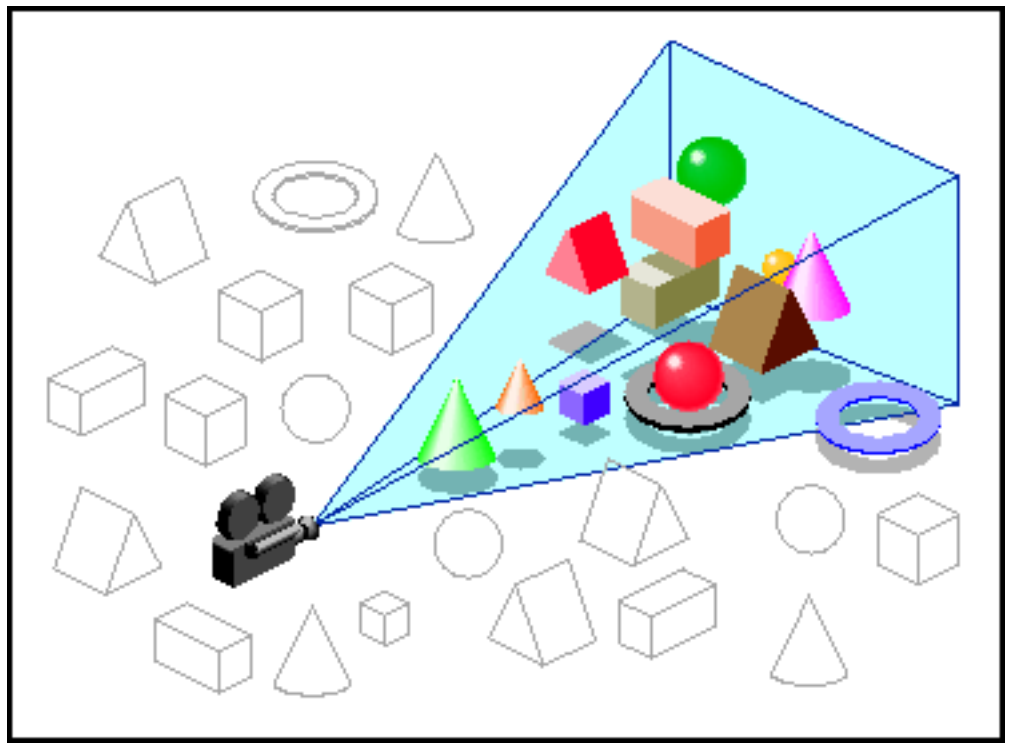
- Update pass (único passo obrigatório)
- Processamento de cena (Informações sobre a câmera estão acessíveis)
- Pós-processamento da cena (Informações sobre a câmera não são mais acessíveis)

Frustum culling



Fonte: http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=0650&db=bks&fname=/SGI_Developer/Optimizer_PG/ch05.html

Frustum culling



Fonte: http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=0650&db=bks&fname=/SGI_Developer/Optimizer_PG/ch05.html

Exemplo de código

Exemplo de aplicação simples usando a engine

main.cpp

(Global Scope)

```
1  #include "math3d/math3d.h"
2  #include "pbge/pbge.h"
3
4  class MySceneInitializer : public pbge::SceneInitializer {
5  public:
6      pbge::SceneGraph * operator () (pbge::GraphicAPI * gfx, pbge::Window * window) { ...
19 private:
20     void configureCamera(pbge::Node * parent, pbge::GraphicAPI * gfx) { ... }
29     void createModel(pbge::Node * parent, pbge::GraphicAPI * gfx) { ... }
53 };
54
55 int main() {
56     pbge::Manager manager;
57     MySceneInitializer sceneInitializer;
58     manager.setWindowTitle("Ellipsoid demo");
59     manager.setWindowDimensions(1024, 768);
60     manager.setSceneInitializer(&sceneInitializer);
61     manager.displayGraphics();
62     return 0;
63 }
```

main.cpp

(Global Scope)

```
1  #include "math3d/math3d.h"
2  #include "pbge/pbge.h"
3
4  class MySceneInitializer : public pbge::SceneInitializer {
5  public:
6      pbge::SceneGraph * operator () (pbge::GraphicAPI * gfx, pbge::Window * window) {
7          pbge::Renderer * renderer = window->getRenderer();
8          renderer->addSceneProcessor(new pbge::RenderPassProcessor);
9          renderer->addPostProcessor(new pbge::BlitToFramebuffer);
10         pbge::Node * root = new pbge::TransformationNode;
11         pbge::SceneGraph * graph = new pbge::SceneGraph(root);
12
13         configureCamera(root, gfx);
14         createModel(root, gfx);
15
16         return graph;
17     }
18
19 private:
20     void configureCamera(pbge::Node * parent, pbge::GraphicAPI * gfx) { ... }
29     void createModel(pbge::Node * parent, pbge::GraphicAPI * gfx) { ... }
53 };
54
55 int main() {
56     pbge::Manager manager;
```

main.cpp

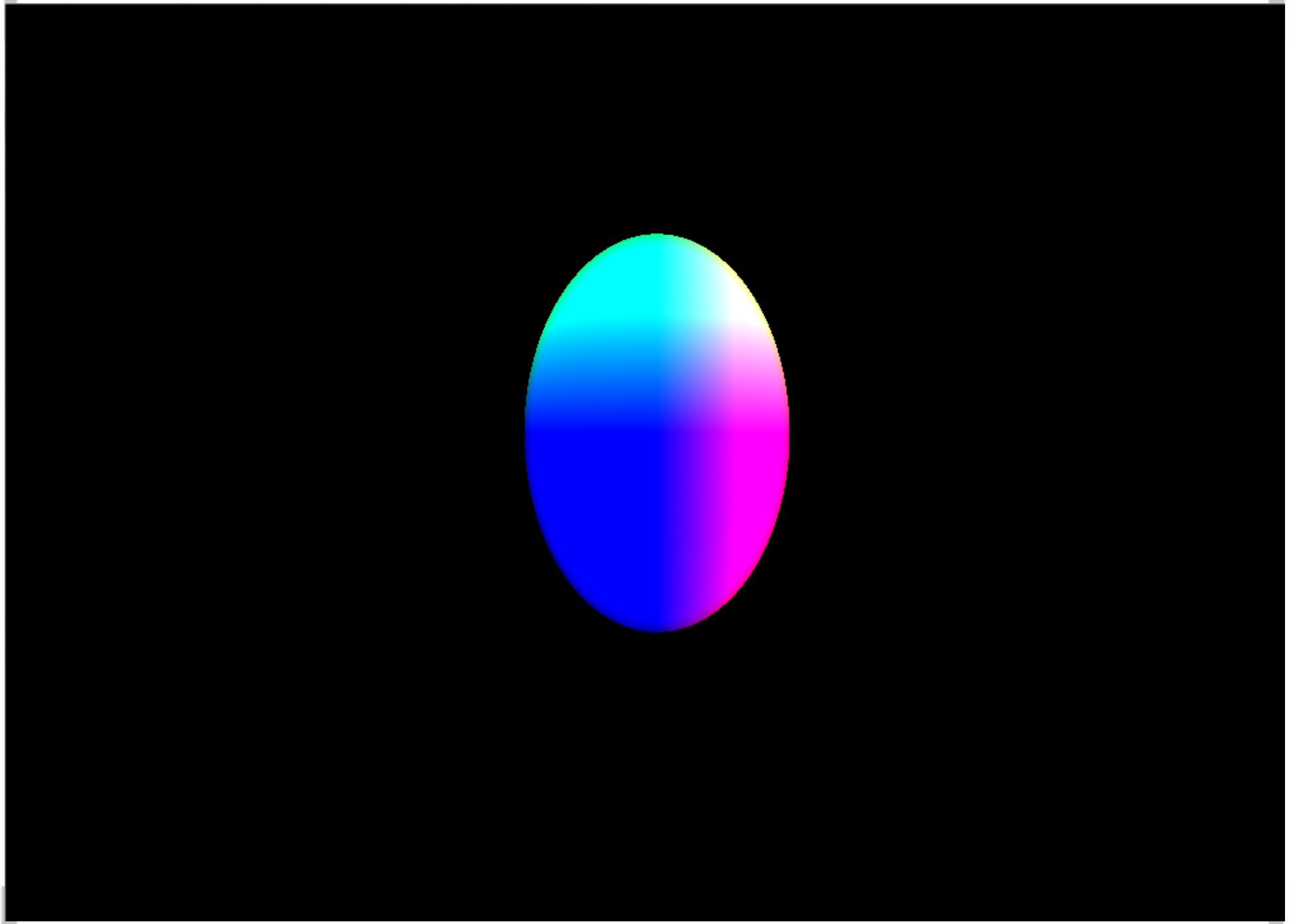
(Global Scope)

```
1  #include "math3d/math3d.h"
2  #include "pbge/pbge.h"
3
4  class MySceneInitializer : public pbge::SceneInitializer {
5  public:
6      pbge::SceneGraph * operator () (pbge::GraphicAPI * gfx, pbge::Window * window) {
19 private:
20     void configureCamera(pbge::Node * parent, pbge::GraphicAPI * gfx) {
21         pbge::TransformationNode * cameraParent =
22             pbge::TransformationNode::translation(0, 0, 10);
23         pbge::CameraNode * camera = new pbge::CameraNode;
24         camera->lookAt(math3d::vector4(0,1,0), math3d::vector4(0, 0, -1));
25         camera->setPerspective(90, 1.0f, 2.0f, 30.0f);
26         cameraParent->addChild(camera);
27         parent->addChild(cameraParent);
28     }
29     void createModel(pbge::Node * parent, pbge::GraphicAPI * gfx) { ... }
53 };
54
55 int main() {
56     pbge::Manager manager;
57     MySceneInitializer sceneInitializer;
58     manager.setWindowTitle("Ellipsoid demo");
59     manager.setWindowDimensions(1024, 768);
60     manager.setSceneInitializer(&sceneInitializer);
```

main.cpp

(Global Scope)

```
20      void configureCamera(pbge::Node * parent, pbge::GraphicAPI * gfx) { ... }
29      void createModel(pbge::Node * parent, pbge::GraphicAPI * gfx) {
30          pbge::VBOModel * sphere = pbge::Geometrics::createSphere(2,100,gfx);
31          pbge::ModelInstance * model = new pbge::ModelInstance(sphere);
32          pbge::GPUProgram * shader = gfx->getFactory()->createProgramFromString(
33              "#version 150\n"
34              "in vec4 pbge_Vertex;\n"
35              "out vec4 color\n;"
36              "uniform mat4 pbge_ModelViewProjectionMatrix;\n"
37              "void main() {\n"
38              "   mat4 scale = mat4(1,0,0,0,\n"
39              "                     0,2,0,0,\n"
40              "                     0,0,1,0,\n"
41              "                     0,0,0,1);\n"
42              "   gl_Position = pbge_ModelViewProjectionMatrix*scale*pbge_Vertex;\n"
43              "   color = vec4(pbge_Vertex.xyz, 1);\n"
44              "}",
45              "in vec4 color;\n"
46              "void main() {\n"
47              "   gl_FragColor = color;\n"
48              "}"
49              );
50          model->setRenderPassProgram(shader);
51          parent->addChild(model);
52      }
```



Campo de grama

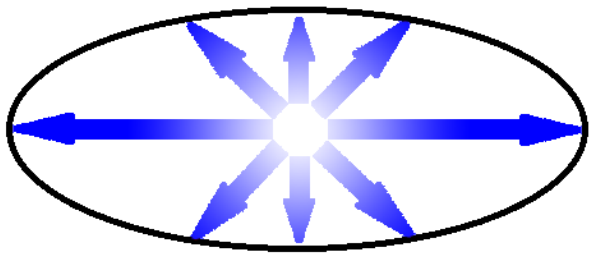
IRM sensíveis a difusão

IRM = Imagem de Ressonância Magnética

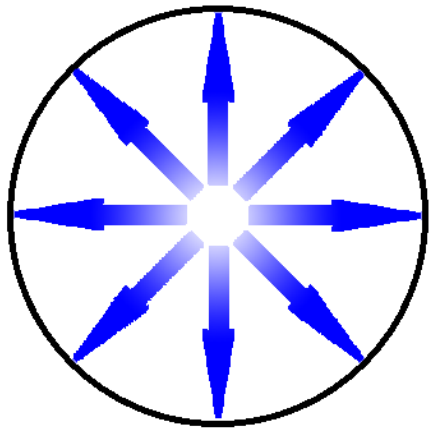
- Imagens em três dimensões
- Difusão da água (tecidos vivos)
- Informações da difusão média da água em cada ponto representada por tensores

Conceitos

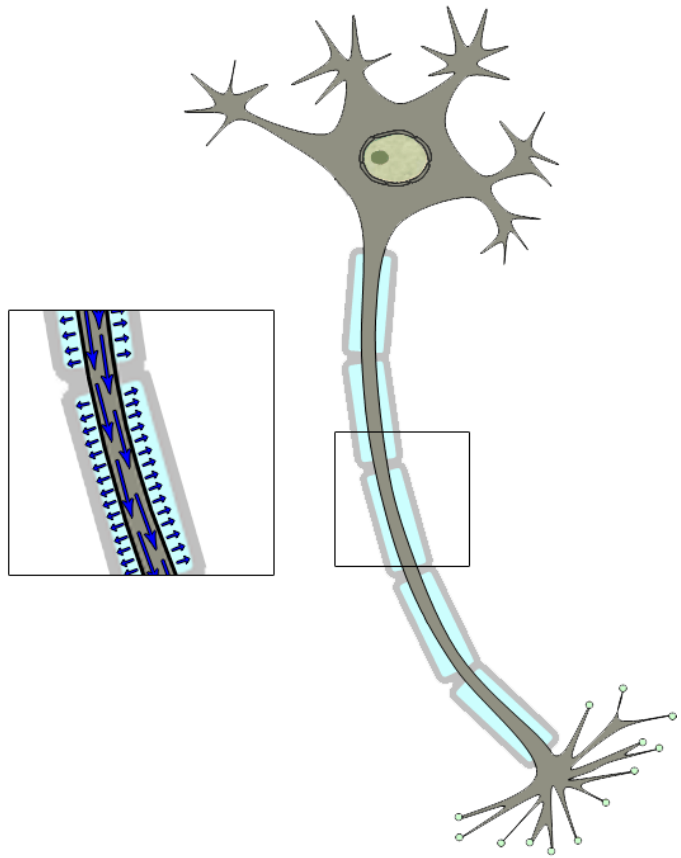
ANISOTRÓPICO



ISOTRÓPICO



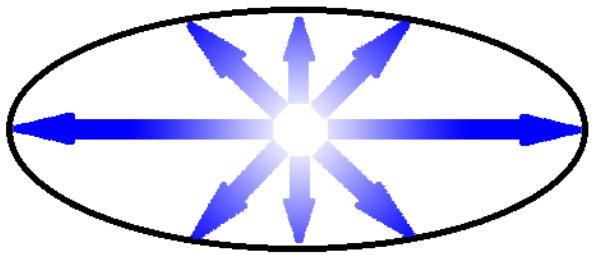
Para que serve?



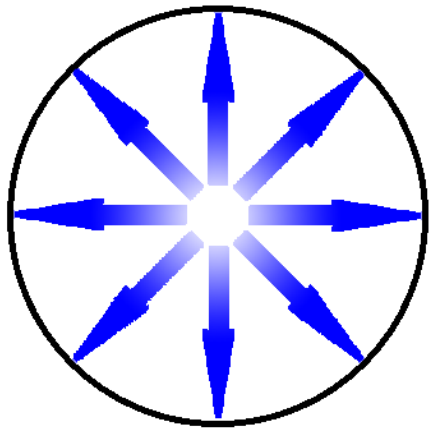
Representação elipsoidal

Autovalores e autovetores do tensor

ANISOTRÓPICO



ISOTRÓPICO

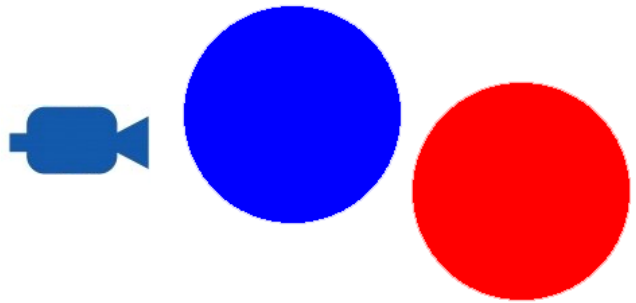


Cérebro

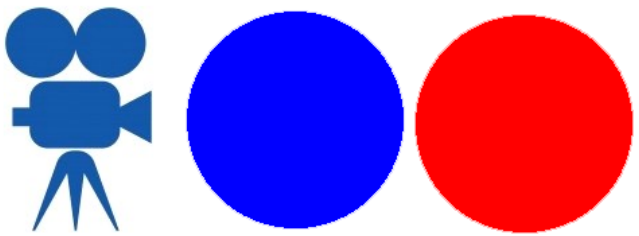
Visualização da representação elipsoidal do campo de tensores de difusão de um cérebro humano

Técnicas - Depth Peeling

VISTA SUPERIOR

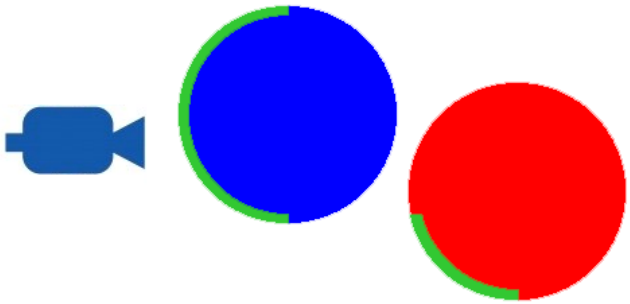


VISTA LATERAL

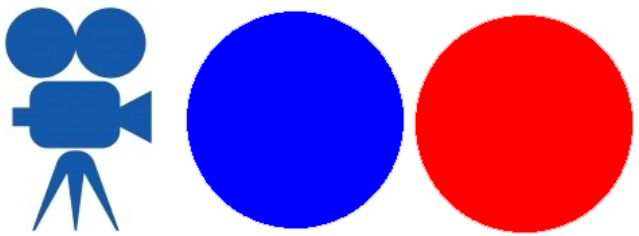


Técnicas - Depth Peeling

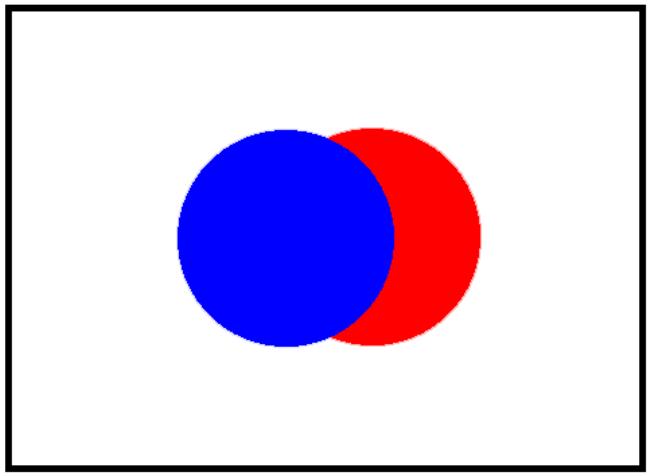
VISTA SUPERIOR



VISTA LATERAL

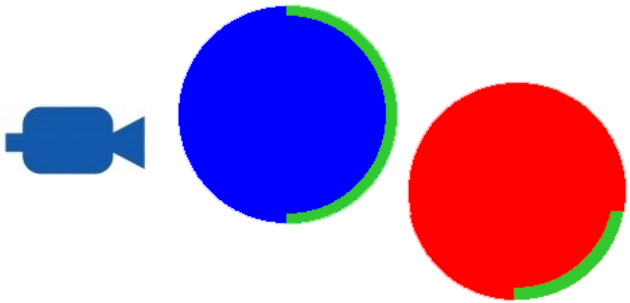


RESULTADO DA ITERAÇÃO

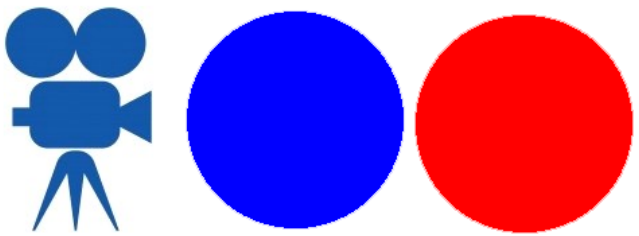


Técnicas - Depth Peeling

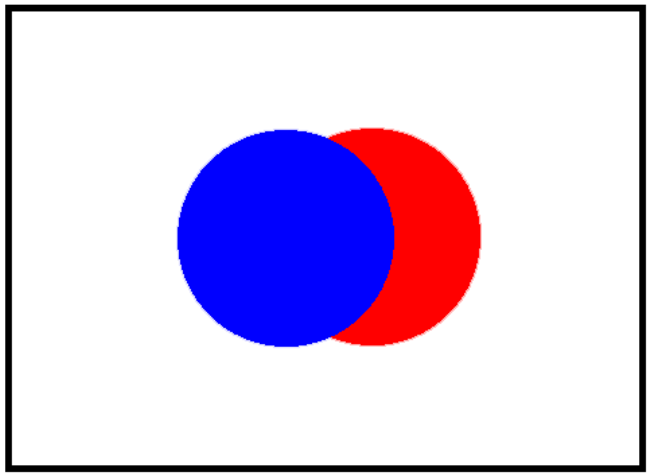
VISTA SUPERIOR



VISTA LATERAL

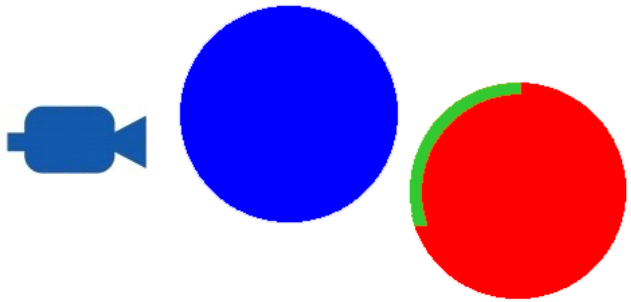


RESULTADO DA ITERAÇÃO

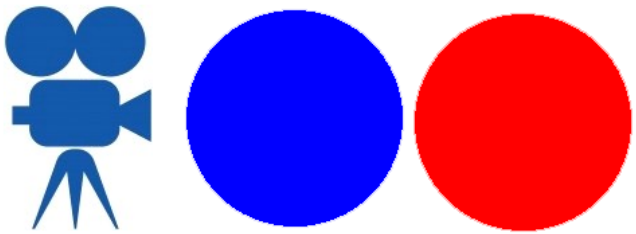


Técnicas - Depth Peeling

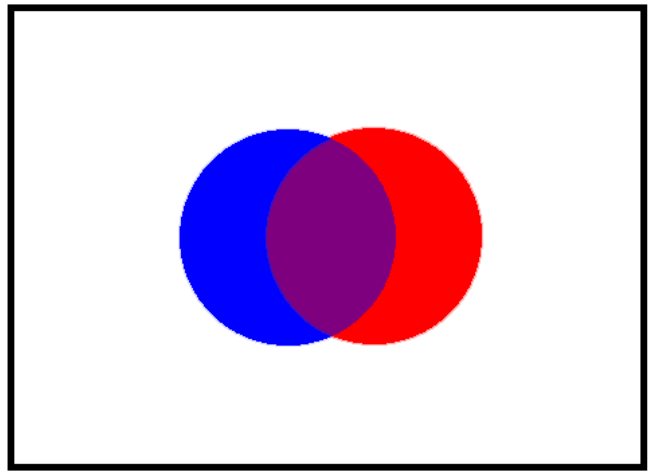
VISTA SUPERIOR



VISTA LATERAL

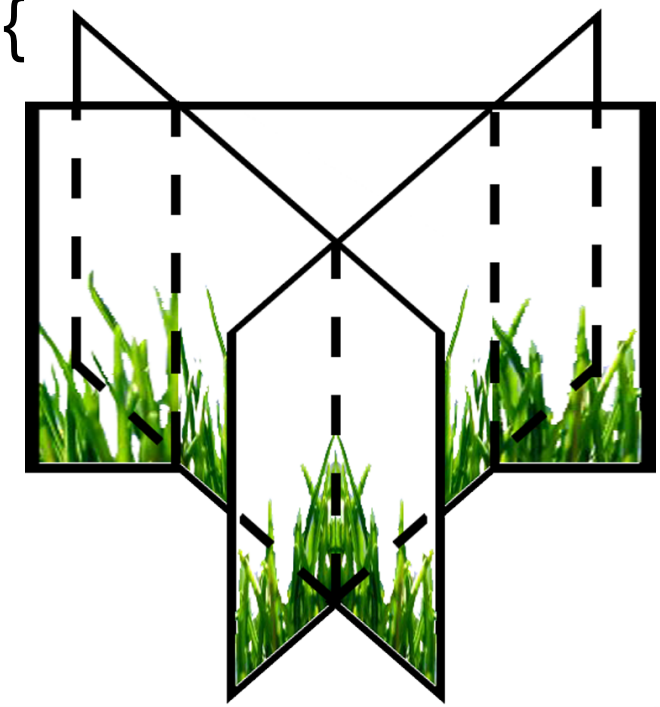


RESULTADO DA ITERAÇÃO



Técnicas - Geometry Instancing

```
model = new Model;
for(id = 0; id < instances; id++) {
    drawInstance(model, id);
}
```



Trabalho futuro

- Aprimorar o framework de shaders para utilizar o conceito de injeção de dependências
- Separar o contexto de renderização da classe GraphicAPI
- Criar proxies para os objetos internos
- Executar o renderizador em uma thread própria

Referências

- cplusplus.com - the c++ resources network. http://www.cplusplus.com/. Acessado em agosto de 2011.
- Peter B. Kingsley. Introduction to diffusion tensor imaging mathematics: Part I. Tensors, rotations, and eigenvectors. Concepts in Magnetic Resonance Part A, 28A(2):101–122, March 2006.
- Opengl 4.1 reference pages. http://www.opengl.org/sdk/docs/man4. Acessado em junho de 2011.
- SHREINER, Dave. **OpenGL Programming Guide.** 7. ed. Addison-Wesley, 2010. 885p.

Dúvidas?

Código disponível em:
https://github.com/victorkendy/PandoraBox

Obrigado!