9/25/2013

OpenGL in MITK

Sandy Engelhardt

dkfz.
DEUTSCHES
KREBSFORSCHUNGSZENTRUM
IN DER HELMHOLTZ-GEMEINSCHAFT

OpenGL (Open Graphics Library)

- cross-language, multi-platform API for rendering 2D and 3D computer graphics
- typically used to interact with a GPU, to achieve hardware-accelerated rendering
- latest OpenGL Version 4.4 released July 22, 2013
- MITK / VTK support OpenGL v. 1.1.

OpenGL naming conventions

1) Prefix: gl or GL
   - function: glFunction()
   - constant: GL_CONSTANT
   - type: GLtype
2) Suffix of function: glFunctionnt()

   n → number of parameter

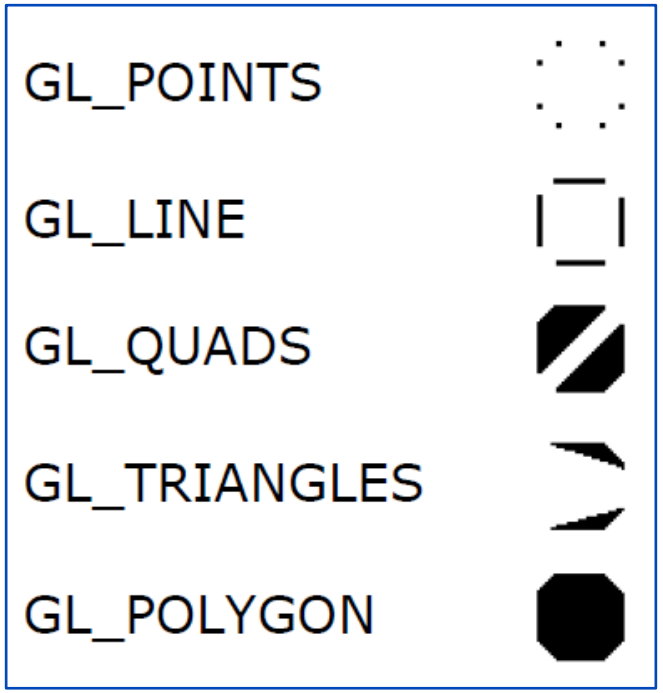
   t → type of parameter

Example: glColor3f()

Rendering Primitives in OpenGL

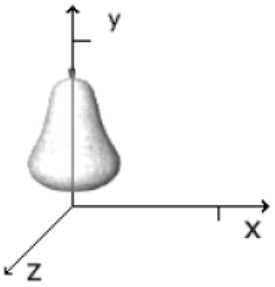
- Draw Primitives:

```
glBegin(GL_PRIMITIV_NAME)
    glVertex3f(x1,y1,z1)
    …
    glVertex3f(xn,yn,zn)
glEnd()
```

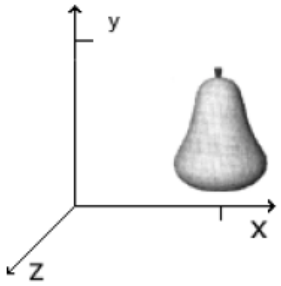


- Concept of statemachine:
  - properties and position of an object depends on the global state
  - individual properties are not assigned to each object

Transformation in OpenGL

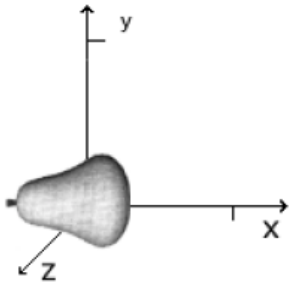


ohne



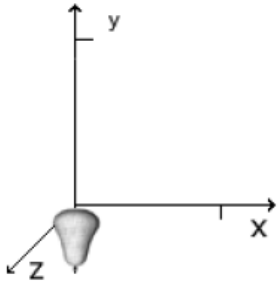
Translation

glTranslate3f (1.0f, 0.0f, 0.0f)



Rotation

glRotate3d (90, 0.0, 0.0, 1.0)



Skalierung

glScaled (0.5, -0.5, 0.5)

```
glTranslate3f(1.0f, 0.0f, 0.0f)
glBegin(GL_PRIMITIV_NAME)
   glVertex3f(x1,y1,z1)
   …
   glVertex3f(xn,yn,zn)
glEnd()
```

Rendering in VTK

- VTK is object oriented, abstracted from OpengGL
- Draw primitives in VTK:

```
vtkSmartPointer<vtkPoints> points = vtkSmartPointer<vtkPoints>::New();
points->InsertNextPoint ( 1.0, 0.0, 0.0 );
…

vtkSmartPointer<vtkTriangle> triangle vtkSmartPointer<vtkTriangle>::New();
triangle->GetPointIds()->SetId ( 0, 0 );
triangle->GetPointIds()->SetId ( 1, 1 );
triangle->GetPointIds()->SetId ( 2, 2 );

vtkSmartPointer<vtkCellArray> triangles = vtkSmartPointer<vtkCellArray>::New();
triangles->InsertNextCell ( triangle );

vtkSmartPointer<vtkPolyData> trianglePolyData = vtkSmartPointer<vtkPolyData>::New();
trianglePolyData->SetPoints ( points );
trianglePolyData->SetPolys ( triangles );

// set mapper …
```

OpenGL vs VTK

Advantages OpenGL

- fast in program execution
- provides more specific solutions
- extensible to accomodate new hardware innovations (additional performance)

Advantages VTK

- easier to understand
- fast implementation

OpenGL Extension Wrangler Library (glew)

- If you want to use a higher OpenGL version, use glew library
  - →enables **OpenGL Extensions** (currently #: 498)
    - GPU vendors provide additional functionalities
    - relax or remove restrictions on existing OpenGL functions
- cross-platform, open-source, C/C++ extension loading library
- provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform
- OpenGL core and extension functionality is exposed in a single header file `<GL/glew.h>`
- http://glew.sourceforge.net

Using OpenGL extensions in VTK (I)

- Include `vtkgl.h` – contains all constants and function required for using OpenGL extensions in a portable and namespace safe way

`vtkOpenGLExtensionManager` –

- Interface class for querying and using OpenGL Extensions

- Get the OpenGL context from the vtkRenderWindow:

```
vtkOpenGLExtensionManager *extensions = vtkOpenGLExtensionManager::New();
extensions->SetRenderWindow(renwin);
```

- Query the vtkOpenGLExtensionManager to see if the extension is supported (valid names see http://www.opengl.org/registry/)

```
if(!extensions->ExtensionSupported("GL_VERSION_1_2")
        || !extensions->ExtensionSupported("GL_ARB_multitexture[1]")) {
        vtkErrorMacro("Required extensions not supported!");
 }
```

[1]ARB = Khronos Group's Architecture Review Board has given explicit approval

Using OpenGL extensions in VTK (II)

- Load extension:

```
extensions->LoadExtension("GL_VERSION_1_2");
extensions->LoadExtension("GL_ARB_multitexture");
```

…

```
*Do*magic*stuff*
```

…

- Naming conventions:
  - constant of an extension: use vtkgl:: instead of GL_
  - function: use vtkgl:: instead of gl

```
vtkgl::ActiveTexture(vtkgl::TEXTURE0_ARB);
```

- Delete them

```
extensions->Delete();
```

Example

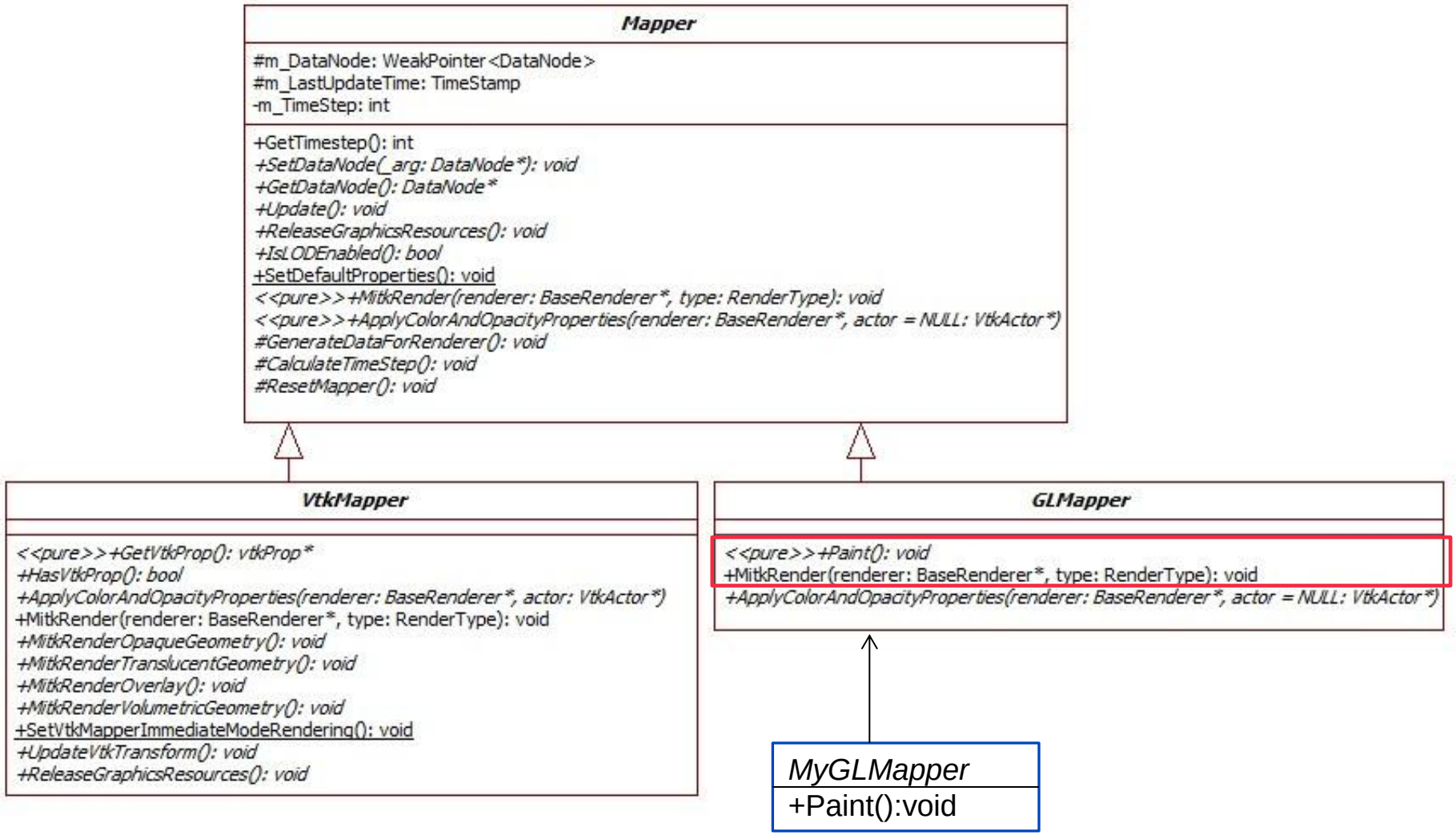
- MITK source /Modules/MITKExt/vtkMitkOpenGLVolumeTextureMapper3D
  - Renders a volume using 3D texture mapping

  - `GL_ARB_vertex_program` (Vertex Shader)
  - `GL_ARB_fragment_program` (Fragment Schader)
  - `GL_EXT_texture3D` (3D texture)
  - `GL_ARB_multitexture` (multiple textures are blended)
  - `GL_ARB_texture_compression`
  - `GL_ARB_texture_non_power_of_two` (texture dimension)

- MBI source /Modules/PathPlanning/mitkVTKOpenGLOcclusionMapper

  - `GL_ARB_occlusion_query`

Mapper Architecture



OpenGL Rendering in MITK