

Projects

Johannes Ibald

November 07, 2020

Contents

1	Resource Latex	1
1.1	A Section	2
1.2	Cool Pictures	3
2		6
3	•	7
4	•	8
5	•	9
6	Creating Realistic Rendering Effects	10
6.1	Understanding graphics shaders	10
6.1.1	osg::Shader	11
6.2	Using uniforms	12
6.2.1	osg::Uniform class	12
6.2.2	Time for Action page 154	13
6.2.3	What just happened?	13
6.3	Working with the geometry shader	14
6.3.1	Time for action - Generating a Bezier curve	14
6.3.2	What just happened?	14
7	Viewing the World	16
7.0.1	From world to screen	16
7.1	The Camera class	18
7.1.1	Rendering order of cameras	20
7.1.2	Time for action creating an HUD camera	21
7.1.3	WTF just happened?	21

Chapter 1

Resource Latex

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. $\sin^2(\alpha) + \cos^2(\beta) = 1$. If you read this text, you will get no information $E = mc^2$. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. $\sqrt[n]{a} \cdot \sqrt[n]{b} = \sqrt[n]{ab}$. This text should contain all letters of the alphabet and it should be written in of the original language. $\frac{\sqrt[n]{a}}{\sqrt[n]{b}} = \sqrt[n]{\frac{a}{b}}$. There is no need for special content, but the length of words should match the language. $a\sqrt[n]{b} = \sqrt[n]{a^n b}$. Hello, here is some text without a meaning. $d\Omega = \sin\vartheta d\vartheta d\varphi$. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. $\sin^2(\alpha) + \cos^2(\beta) = 1$. This text should contain all letters of the alphabet and it should be written in of the original language $E = mc^2$. There is no need for special content, but the length of words should match the language. $\sqrt[n]{a} \cdot \sqrt[n]{b} = \sqrt[n]{ab}$. Hello, here is some text without a meaning. $\frac{\sqrt[n]{a}}{\sqrt[n]{b}} = \sqrt[n]{\frac{a}{b}}$. This text should show what a printed text will look like at this place. $a\sqrt[n]{b} = \sqrt[n]{a^n b}$. If you read this text, you will get no information. $d\Omega = \sin\vartheta d\vartheta d\varphi$. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should

match the language. $\sin^2(\alpha) + \cos^2(\beta) = 1$. Hello, here is some text without a meaning $E = mc^2$. This text should show what a printed text will look like at this place. $\sqrt[n]{a} \cdot \sqrt[n]{b} = \sqrt[n]{ab}$. If you read this text, you will get no information. $\frac{\sqrt[n]{a}}{\sqrt[n]{b}} = \sqrt[n]{\frac{a}{b}}$. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. $a\sqrt[n]{b} = \sqrt[n]{a^n b}$. This text should contain all letters of the alphabet and it should be written in of the original language. $d\Omega = \sin\vartheta d\vartheta d\varphi$. There is no need for special content, but the length of words should match the language. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. $\sin^2(\alpha) + \cos^2(\beta) = 1$. If you read this text, you will get no information $E = mc^2$. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. $\sqrt[n]{a} \cdot \sqrt[n]{b} = \sqrt[n]{ab}$. This text should contain all letters of the alphabet and it should be written in of the original language. $\frac{\sqrt[n]{a}}{\sqrt[n]{b}} = \sqrt[n]{\frac{a}{b}}$. There is no need for special content, but the length of words should match the language. $a\sqrt[n]{b} = \sqrt[n]{a^n b}$.

1.1 A Section

Hello, here is some text without a meaning. $d\Omega = \sin\vartheta d\vartheta d\varphi$. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. $\sin^2(\alpha) + \cos^2(\beta) = 1$. This text should contain all letters of the alphabet and it should be written in of the original language $E = mc^2$. There is no need for special content, but the length of words should match the language. $\sqrt[n]{a} \cdot \sqrt[n]{b} = \sqrt[n]{ab}$. Hello, here is some text without a meaning. $\frac{\sqrt[n]{a}}{\sqrt[n]{b}} = \sqrt[n]{\frac{a}{b}}$. This text should show what a printed text will look like at this place. $a\sqrt[n]{b} = \sqrt[n]{a^n b}$. If you read this text, you will get no information. $d\Omega = \sin\vartheta d\vartheta d\varphi$. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

$$\sin^2(\alpha) + \cos^2(\beta) = 1.$$

- First item in a list
- Second item in a list
- Third item in a list
- Fourth item in a list
- Fifth item in a list

1. First item in a list
2. Second item in a list
3. Third item in a list
4. Fourth item in a list
5. Fifth item in a list

First item in a list

Second item in a list

Third item in a list

Fourth item in a list

Fifth item in a list

1.2 Cool Pictures

Figure 1.1: Itsuki heard some shocking information

Hello, here is some text without a meaning $E = mc^2$. This text should show what a printed text will look like at this place. $\sqrt[n]{a} \cdot \sqrt[n]{b} = \sqrt[n]{ab}$. If you read this text, you will get no information. $\frac{\sqrt[n]{a}}{\sqrt[n]{b}} = \sqrt[n]{\frac{a}{b}}$. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. $a\sqrt[n]{b} = \sqrt[n]{a^n b}$. This text should contain all letters of the

alphabet and it should be written in of the original language. $d\Omega = \sin\vartheta d\vartheta d\varphi$. There is no need for special content, but the length of words should match the language. Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. $\sin^2(\alpha) + \cos^2(\beta) = 1$. If you read this text, you will get no information $E = mc^2$. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. $\sqrt[n]{a} \cdot \sqrt[n]{b} = \sqrt[n]{ab}$. This text should contain all letters of the alphabet and it should be written in of the original language. $\frac{\sqrt[n]{a}}{\sqrt[n]{b}} = \sqrt[n]{\frac{a}{b}}$. There is no need for special content, but the length of words should match the language. $a\sqrt[n]{b} = \sqrt[n]{a^n b}$.

I Material

- Fichte ?

II Deskplate

- amount: 1
- measurements: 2500 x 1250 x 27 [mm]

III Reinforcement

- amount: 1
- Length: 2500 mm
- Width: ≤ 90 mm
- Height: $160 > H > 140$ [mm]
- width (W) is determined by two Ikea Alex but proper legs might give more flexibility later on

Chapter 2

Chapter 3



Chapter 4



Chapter 5



Chapter 6

Creating Realistic Rendering Effects

6.1 Understanding graphics shaders

1. OpenGL shading language (GLSL) provides the ability to develop graphics shaders
→ blocks of graphic software instructions to calculate more realistic rendering effects, rather than fixed function states.
2. steps to designing shaders and applying them to a scene
 - write your own shaders ("like C programs"). They are treated as a set of strings passed to the hardware so create them on the fly or read them as text files.
 - specify no more than a vertex shader, a geometry shader and a fragment shader to be processed in the OpenGL pipeline. Each stage has only one `main()` function.
 - will totally replace fixed functionalities such as fog, lighting and texture mapping, which have to be re-implemented in your shader source code.
 - Shaders require OpenGL API to compile and execute them.
 - Vertex shader can apply transformations to each vertex
 - Fragment shaders calculate the color of individual pixels coming from the rasterizer;
 - Geometry shaders re-generate geometries from existing vertices and primitive data

6.1.1 osg::Shader

- define shader object containing source code strings.
- `setShaderSource()` specifies the src code text from a `std::string` variable
- `loadShaderSourceFromFile()` reads a source file from drive.
- construct shader object from existing string like this:

```
osg::ref_ptr<osg::Shader> vertShader =
    new osg::Shader( osg::Shader::VERTEX, vertText );
```

- input param `OSG::Shader::VERTEX` represents the vertex shader. Use `GEOMETRY` or `FRAGMENT` enums instead to specify geometry- or fragment shader.

```
osg::ref_ptr<osg::Shader> fragShader =
    new osg::Shader( osg::Shader::FRAGMENT, fragText );

osg::ref_ptr<osg::Shader> geomShader =
    new osg::Shader( osg::Shader::GEOMETRY );

geomShader -> loadShaderSourceFromFile( "source.geom" );
```

→source.geom contains geometry shader.

- `osgDB::readShaderFile()` may be even better
 - automatically checks shader types (via extensions: .vert, .frag, .geom)
 - returns `osg::Shader` instance of correct type and data:

```
osg::Shader* fragShader =
    osgDB::readShaderFile( "source.frag" );
```

→shaders are set and ready to be use

→use `osg::Program` calss and `addShader()` method to include include shaders and set GLSL rendering attribute and modes to a state set.

- most other fixed-function states willbecome incalid after the shaders make effects, including lights, materials, fog, texture mapping, texture coordinate generation and texture environment.
- following code adds all above shaders to an `osg::Program` object and attaches it to the state set of existing node:

```

osg::ref_ptr<osg::Program> program =
    new osg::Program;
program -> addShader( vertShader.get() );
program -> addShader( fragShader.get() );
program -> addShader( geomShader.get() );
node -> getOrCreateStateSet() -> setAttributeAndModes(
    program.get() );

```

6.2 Using uniforms

- three types of inputs and outputs in a typical shader:
 - uniforms
 - vertex attributes
 - varyings
- Uniforms and Vertex Attributes are read-only during the sahder's execution, but can be set by host OpenGL or OSG apps.
 - They are actually global GLSL variables used for interactions between shaders and user applications.
- Varyings are used for passing data from one shader to the next one
 - tehy are invisible to external programs
- OSG uses osg::Uniform class

6.2.1 osg::Uniform class

- used to define a SLGL uniform variable
- constructor has a name and initial value param, which should match the definition in the shader souce code, e.g:

```

float length = 1.0f;
osg::ref_ptr<osg::Uniform> uniform =
    new osg::Uniform( "length", length );

```

- add uniform object to state set, which has attached osg::Program object via addUniform():

```

stateset -> addUniform( uniform.get() );

```

There should be a variable defined in one of the shader sources:

```
uniform float length;
```

Otherwise, uniform variable will not be available in either OSG programs or shaders.

- Uniforms can be any basic type, or any aggregation of types, such as Boolean, float, integer, 2D/3D/4D vector, matrix and various texture samplers.
- `osg::Uniform` class accepts all basic types with constructor and `set()` method.
→ additionally, `osg::Matrix2` and `osg::Matrix3`
- to bind texture sampler (used in shaders to represent a particular texture) you specify the texture mapping unit by using an unsigned int:

```
osg::ref_ptr<osg::Uniform> uniform =  
    new osg::Uniform( "texture", 0 );
```

- there must already be an `osg::Texture` object at unit 0, as well as a sampler uniform in the shader source:

```
uniform sampler2D texture;
```

→ assume that it's a 2D texture that will be used to change the shader's executing behavior.

6.2.2 Time for Action page 154

6.2.3 What just happened?

basic algorithm for caroon shading:

- if there's a normal that is close to the light direction, the brightest tone
→ `color1` is used.

- as the angle between light direction and surface normal is increasing
→darker tones will be used (color2, color3, color4)
→provides an intensity value for selecting tones.
→all four tones are declares as 4D vectors in FRAGMENT SHADER and passed to osg::Uniform objects as osg::Vec4 variables in the user app.

6.3 Working with the geometry shader

- geometry shader is included into the OpenGL 3.2 core
→in lower versions it is udes as an extension (GL_EXT_ geometry_shader4) which should be declared in the shader sourve code.
- geometry shader has new sdjacency primitives
→can be used as arguments of osg::PrimitiveSet derived classes.
→also requires setting up params in order to maipulate the shader operations:

1. GL_GEOMETRY_VERTICES_OUT_EXT: numns of vertices that the shader will emit
2. GL_GEOMETRY_INPUT_TYPE_EXT: the primitive type to be sent to the shader
3. GL_GEOMETRY_OUTPUT_TYPE_EXT: primitive type to be emitted from the shader

→osg::Program class's setPatameter() sets values for these params

→100 vertices will be emitted from the shader to the primitive assembly processor in the rendering pipeline:

```
program -> setParameter( GL_GEOMETRY_VERTICES_OUT_EXT, 100 );
```

6.3.1 Time for action - Generating a Bezier curve

P158

6.3.2 What just happened?

- geometry shader defines a new primitive type GL_LINE_STRIP_ADJACENCY_EXT which means a line strip with adjacency
→first and last vertices provide adjacency information bur aren't visible as line segments.

→thus we can use these two extra vertices as the endpoints of a Bezier curve and the others as control points

→that is actually what we read from the GLSL variable `gl_Position[0]` to `gl_PositionIn[3]`.

- Cubic Bezier curve equation:

$$P(t) = (1-t)^3 * P0 + 3*t*(1-t)^2 * (1-t) * P2 + t^3 * P3 \text{ with } 0 \leq t \leq 1$$

See summary.

Chapter 7

Viewing the World

Focus:

- understanding the coordinate system defined in OpenGL
- alternating the view point and orientation, projection frustum, and final viewport
- changing and controlling the rendering order if there exists more than one camera
- how to create single and composite viewers
- how to manage global display settings and generate easy-to-use stereo visualization effects
- how to apply the rendered scene as a texture object - so called rendering to textures (RTT)

7.0.1 From world to screen

this subsection will be shorter, since a version of this is already in my personal notebook.

modelmatrix

used to describe the specific location of an object in the world.

→ transforms object's local coord sys to world coord sys. Both coord. systems are right-handed.

view matrix

→transforms entire world into view space. suppose we have a camera placed at a certain position in the world; the inverse of the camera's transformation matrix is actually used as the view matrix.

In the right-handed view coord sy, OpenGL defines that the camera is always located at the origin (0, 0, 0), and facing towards the negative Z axis.

→Hence, we can represent the world on our camera's screen.

Note:

There is no separate model matrix or view matrix in Open GL.

→however, it defines a model-view matrix to transform from the object's local space to view space, which is a combination of both matrices.

→to transform vertex V in local space to V_e in view space, we have:

$$V_e = V * modelViewMatrix$$

projection matrix

we have to:

- determine how 3D objects are projected onto the screen (perspective or orthogonal)
- calculate the frustum.

→Projection matrix is used to specify the frustum in the world coordinate system with six clipping planes: left, right, bottom, top, near and far planes.

→OpenGL function: `gluPerspective()`, determines a field of view with camera lens params.

- resulting coord sys is called: Normalized Device Coordinate System
 - it ranges from -1 to +1 in each of the axes.
 - is changed to left-handed now.
- as a final step:
 - project all result data onto viewport. (the window)
 - define the window rectangle in which the final image is mapped
 - As well as Z Value of the window coordinates.
- Now the 3D scene is rendered to a rectangular area on your 2D screen.

MVPW matrix

Finally, the screen coord V_s can represent the local vertex V in the 3D world by using the so called MVPW matrix:

$$V_s = V * modelViewMatrix * projectionMatrix * windowMatrix$$

The V_s is still a vector that represents a 2D pixel location with a depth value.

By reversing this mapping process, we can get a line in the 3D space from a 2D screen point (X_s, Y_s)

→that's because th 2D point can actually be treated as two points: one on the near clipping plane ($Z_s = 0$) and the other on the far plane ($Z_s = 1$).

The inverse of the MVPW matrix is used to obtain the result of the "unproject" work:

$$V0 = (X_s, Y_s, 0) * invMVPW$$

$$V1 = (X_s, Y_s, 1) * invMVPW$$

7.1 The Camera class

- it's popular to use `glTranslate()` and `glRotate()`
→moves the scene
- it's popular to use `gluLookAt()`
→moves the camera
- though they are all replaceable by `glMultMatrix()`
→in fact, these functions do the same thing: calculate the model-view matrix for transforming data from world space to view space.
- similarly, OSG had `osg::Transform` class
→adds or sets its own matrix to the current model-view matrix when placed in the sg
- BUT: we always intend to operate on model matrix by using the
→`osg::MatrixTransform` and `osg::PositionAttitudeTransform` subclasses
→we handle the view matrix with the `osg::Camera` subclass.
- `osg::Camera` class is one of the most important classes in the core OSG libraries.
→can be used as Group node

- but it is far more than a common node
 - main functionalities in four categories:
1. `osg::Camera` class handles the view matrix projection matrix and viewport
 - affects all its children and project them onto the screen
 - Related methods:
 - public: `setViewMatrix()` and `setViewMatrixAsLookAt()` methods set the view matrix by using the `osg::Matrix` variable or classic eye/center/up variables.
 - public `setProjectionMatrix()` method accepts an `osg::Matrix` parameter in order to specify the projection matrix
 - other convenient methods:
 - `setProjectionMatrixAsFrustum()`
 - `setProjectionMatrixAsOrtho()`
 - `setProjectionMatrixAsOrtho2D()`
 - `setProjectionMatrixAsPerspective`
 are used to set a perspective or orthographic projection matrix with different frustum parameters.
 they work just like the OpenGL projection functions (... , see page 165)
 - public `setViewport()` method defines a rectangular window area with an `osg::Viewport` object.

set view and projection matrix of a camera node, set its viewport to $(x, y) - (x + w, y + h)$:

```
camera -> setViewMatrix( viewMatrix );
camera -> setProjectionMatrix( projectionMatrix );
camera -> setViewport( new osg::Viewport( x, y, w, h ) );
```

Obtain current view and projection matrices and viewport of the `osg::Camera` object by using the corresponding `get*()` methods at any time, e.g.:

```
osg::Matrix viewMatrix = camera -> getViewMatrix();
```

get position and orientation of view matrix:

```
osg::Vec3 eye, center, up;
camera -> getViewMatrixAsLookAt( eye, center, up );
```

2. `osg::Camera` encapsulates the OpenGL functions, such as `glClear()`, `glClearColor()`, and `glClearDepth()`, and clears the frame buffers and presets their values when redrawing the scene to the window

in every frame.

Primary methods include:

- `setClearMask()` method, sets buffer to be cleared.
default:

`GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT`

- `setClearColor()` method sets the clear color in RGBA format, by using an `osg::Vec4` variable.
 - similarly there's `setClearDepth()`, `setClearStencil()`, `setClearAccum()` (and their `get*()` methods)
3. third category includes the management of OpenGL graphics context associated with this camera.
→Chapter 9 Interacting with Outside Elements
 4. Finally, a camera can attach a texture object to internal buffer components (color buffer, depth buffer, and so on) and directly render the sub-scene graph into this texture.
→the resultant texture can then be mapped to surfaces of other scenes. This technique is named render-to-textures or texture baking
→later this chapter.

7.1.1 Rendering order of cameras

- at least one main camera node in any sg.
→created and managed by the `osg::Viewer` class
→read it with `getCamera()` method.
- It automatically adds the root node as its child node before starting the simulation.
→by default all other cameras (directly and indirectly added to root node) will share the graphics context associated with the main camera, will share the graphics context associated with the main camera + draw their own sub-scenes successively onto the same rendering window.
- `osg::Camera` class provides `setRenderOrder()` method to precisely control the rendering order of cameras.
→It has an order enum and an optional order num param.
→first enum is either `PRE_RENDER` or `POST_RENDER` (indicates general rendering order)
→second is an interger num for sorting cameras of the same type in ascending order. (default = 0)

- this will force OSG to render camer1 first, then camera2 (larger int num), then camera3:

```
camera1 -> setRenderOrder( osg::Camera::PRE_RENDER );
camera2 -> setRenderOrder( osg::Camera::PRE_RENDER, 5 );
camera3 -> setRenderOrder( osg::Camera::POST_RENDER );
```

If a camera is rendered first (**PRE_RENDER**) it's rendering result in the buffers will be cleared and covered by the next camera, and the viewer may not be able to see its sub-scene. This is especially useful in the case of the render-to-textures process, because we want the sub-scene to be hidden from the screen, and update the attached texture objects before starting the main scene.

In addition, if a camera is rendered afterwards (**POST_RENDER**), it may erase the current color and depth values in the buffers.
→ avoid this by calling `setClearMask()` with fewer buffer masks. (HUD head up display)

7.1.2 Time for action creating an HUD camera

P168

7.1.3 WTF just happened?

an additional camera contains the glider model that is to be rendered as its sub-scene-graph on top of the rendering result (color buffer and depth buffer) of the main camera.

The additional camera's goal is to implement a HUD scene that overlays the main scene. It clears the depth buffer to ensure that all pixel data drawn by this camera can pass the depth test. However, the color buffer is not cleared, keeping the uncovered pixel data of the main scene on the screen. That is why we set it up like this;

```
camera -> setClearMask( GL_DEPTH_BUFFER_BIT ); // no color buffer bit
```