Projects

Johannes Ibald

November 07, 2020

Contents

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. $\mathrm{d}\Omega = \sin\vartheta \mathrm{d}\vartheta \mathrm{d}\varphi$. 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. $\mathrm{d}\Omega = \sin\vartheta \mathrm{d}\vartheta \mathrm{d}\varphi$. 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. $a\sqrt[n]{b}=\sqrt[n]{a^nb}$. This text should contain all letters of the alphabet and it should be written in of the original language. $\mathrm{d}\Omega=\sin\vartheta\mathrm{d}\vartheta\mathrm{d}\varphi$. 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. $a\sqrt[n]{b}=\sqrt[n]{a^nb}$.

1.1 A Section

Hello, here is some text without a meaning. $\mathrm{d}\Omega=\sin\vartheta\mathrm{d}\vartheta\mathrm{d}\varphi$. 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. $a\sqrt[n]{b}=\sqrt[n]{a^nb}$. If you read this text, you will get no information. $\mathrm{d}\Omega=\sin\vartheta\mathrm{d}\vartheta\mathrm{d}\varphi$. 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. $\mathrm{d}\Omega = \sin\vartheta \mathrm{d}\vartheta \mathrm{d}\varphi$. 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 relistic rendering effects, rather than fixed function states.

2. steps to desing shaders and applying them to a sg

   - 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 scan apply transformations to each vertex
   - Fragment shaders calculate the color of infividual 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 objectand 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 exevution, 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 SLSL uniform cariable

- 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 cariable will not be availabel 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 samplet 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 alorithm 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`: nums 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 wil 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 vertives 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:
  $P(t) = (1-t)^3 * P0 + 3 * t * (1-t)^2 * (1-t) * P2 + t^3 * P3 \; with \; 0 \leq t \leq 1$

See summary.

Chapter 7

Viewing the World

Focus:

- understandig 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 dispay 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 vertain 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 Ve in view space, we have:
$Ve = V * modelViewMatrix$

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 axed.
  →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 Vs can represent the local vertex V in the 3D world by using the so called MVPW matrix:

$$Vs = V * modelViewMatrix * projectionMatrix * windowMatrix$$

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

The inverse of the MVPW matrix is used to obtain the result of the "unproject" work:
$V0 = (Xs,\ Ys,\ 0)\ *\ invMVPW$
$V1 = (Xs,\ Ys,\ 1)\ *\ invMVPW$

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 bes 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 viewpoert
     →affects all its chilfren 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 perxpevtive 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::Viewpoert 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 viewpoert of the osg::Camera object by using the correspoinding get*() methods at any time, e.g.:

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

     get position and orientation of view matrix:

     ```
     osg::Vec3 eye, venter, 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(), setClelarStencil(), setClear-Accum() (and their get*() methods)

3. third category includes the management of OpenGL graphgics context associated with this camera.
   →Chapter 9 Interacting with Outside Elements

4. Finally, a camera can attach a texture object to internal buffer coponents (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 techique 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::ViewerViewer 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 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 especaially 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 maks.(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
```

7.1.4 using a single viewer

OSG supports the single viewer class osgViewer::Viewer for holding a view on a single scene.
setSceneData() method
→manages the scene graph's root node

run()
→starts the simulation loop (scene is rendered per frame) →the frame buffer is updated continuously by the result of every rendering cycle (-¿ frame)

the viewer also contains an osg::Camera object as the main camera.
View Matrix of the camera is controlled by the viewer's internal osgGA::CameraManipulator object.

User input events are received and handled by the viewer as well..
→this works via a list of osgGA::GUIEventHandler handlers.

The viewer can even be set up in full screen mode, in a window and onta a spherical display.

7.1.5 Digging into the simulation loop

The simularion loop defined by the run() method always has three types of tasks to perform:

1. specify the main camera's manipulator
2. set up associated graphics contexts
3. render frames in cycles

The manipulator can read keyboard and mouse events and accordingly adjust the main camera's view matrix to navigate the scene graph.
→set by using setCameraManipulator() method
→param: osgGA::Cameramanipulator subclass
e.g.:

```
viewer.setCameraManipulator( new osgGA::TrackballManipulator );
```

→adds trackball(arc ball) manip to viewer object, (free motion behavious)
→because camera manipulator is kept a smart pointer in the viewer, we can assign a new manip by using the setVameraMAnipulator() method any time.
see page 170 for table with maipulators
→beware , to declare and use a manip you should add the osgGA library as a dependence of your project
→CMake scripts
the graphics contexts of a viewer, as well as possible threads and resourfes, are all initialized ni the realize(0 method
→automatically called before the first frame is rendered
Now the viewer enters the loop:

→each time frame() method is used to render a frame. It checks if the rendering process should stop and exit with the don() method. The process can be described with just a few lines of code:

```
while( !viewer.done() )
{
        viewer.frame();
}
```

→default rendering scheme used in the viewer class.
Frame rate is synched to the monitor's refresh rate to avoid wasting system energy, if the vsync option of the gpu is on.
OSG supports another on-demand rendering scheme:

```
viewer.setRunFrameShceme( osgViewer::Viewer::ON_DEMAND );
```

noe the frame() method will only be executed when there are scene graph mods, updating processes, or user input events, until the scheme is changed back to the default value of CONTINUOUS.

As an addition, the osgViewr::Viewer class also contains a setRunMaxFrameRate() method which uses a frame rate number as the param.
→can set a max frame rate
→controls viewer running to force rendering frames without lots of consumption.

7.1.6 Time for action - custom simulation loop

run() was used many times.
→performed update, cull ad draw traversals each frame.
→see P172.

7.1.7 What on earth just happended?

this was the concept of pre- and post-frame events and assume they are executed before and after frame() method.
→inaccurate.

multiple threads are used to manage user updating, culling and drawing of different cameras.
→especially with multiple screens, processors, and gpu.
frame() method only starts a new updating/culling/drawing traversal work, but does not take care of thread synchronization. In this case, the code before and after frame() will be considered unstable and unsafe, bevause they may

conflict with other process threads when reading or writing the scene graph.
→so the approach described here is not recommended for future development.
→"correct" methods in next chapter.

When will the viewer.done() return true?
Of course, you can set the done flag via setDone() meghod of viewe OSG system will check if all present graphics contexts (for example, the rendering window)have been closed, or if the Esc key is pressed which will also change the done flag.
setkeyEvenetSetsDone() method can even set which key is going to carry out the duty instead of the default Esc (osr set this to 0 to turn off the feature).

7.1.8 Using a composite viewer

osgViewr::VIewer class manages only one single view on one scene graph.
osgViewer::CompositeViewer class supports multiple views and multiple scenes.
This has the same methods such as run(), frame() and done(0 to manage the rendering process, but also supports adding and removing independent scene views by using the addView() and removeView() meghods, and obtaining a view object at a specific index by using the getView() method. The view object here is defined by the osgViewer::View class.
osgViewer::View class is the super class of osgViewer::Viewer
→it accepts setting a root node as the scene data, and adding a camera manipulator and event handlers to make use of user events as well.
Ther main difference between osg::Viewer::View and osgViewer::Viewer is that the former cannot be used as a single viewer directly - that is, it doesn't have run() or frame() methods.

To add a created view object to the composite viewer:

```
osgViewer::CompositeViewer multiviewer;
multiviewer.addView( view );
```

7.1.9 Time for action - rendering more scenes at one time

Multi-viewers are practical in representing comlex scenes, for instance, to render a wide area with a main view and an eagle eye view, or to display the front, side, top, and persepective views of the same scene. Here we eill create three separate windows, containintg three different models, each of which can be independently manipulated.

7.1.10 What the fuck just happened?

it's possible to create three osg::Camera nodes, add different sub-scenes to them, and attach them to different graphics contexts (rendering window) in order to achieve the same result as the previous image.
→every osgViewer::View object has an osg::Camera node that can be used to manage its subscene and associated window.
→it actually works like a container.

However, the osgViewer::View class handles manipulator and user events, too.
→in a composite viewer, each osgViewer::View object holds its own manipulator and event handlers (this will be siscussed in Chapter 9, Interacting with Outside Elements).
However, a set of cameras can hardly interact with user inputs separately. That is why we choose to use a composite viewer and a few view objects to represent multiple scenes in some cases.

7.1.11 Changing global display settings

OSG manages a set of global display settings that are required by cameras, viewers, and other scene elements. It uses the singleton pattern to declare a unique instance of the container of all of these settings, by using the osg::DisplaySettings class. We can thus obtain the display settings instance at any time in our apps:

```
osg::DisplaySettings* ds = osg::DisplaySettings::instance();
```

The osg::DisplaySettingsa instance sets up properties requested by all newly created rendering devices, ainly OpenGL graphics contexts of rendering windows. Its chracteristics include:

1. setDoubleBuffer() method: set double or single buffering. Default is on.

2. setDepthBuffer() method: whether to use depth buffer or not. Default is on.

3. setMinimumNumAlphaBits() (and others): set bits for an OpenGL alpha buffer, a stencil buffer, accumulation buffer. Defaults are all 0.

4. setNumMultiSamples(): set using multisampling buffers and number of samples. default is 0.

5. enable stereo rendering and configure stereo mode and eye mapping parameters

→some of these characteristics can be separately set for different graphics contexts by sing a specific traits structure. For now we use global display settings.

7.1.12 Time for action - enabling global multisampling

P180

7.1.13 wth just happened?

multisampling allows apps to create a frame buffer with a given number of samples per pixel.
→contains color depth, stencil info.
→more video memory is required but a better rendering result will be produced.
OSG has an internal graphics context manager osg::GraphicsContext:
→it's subclass osg::GraphicsWindowWin32 (look up Linux version) manages the config and creation of rendering windows under Windows.
It will apply these two attributes to the encapsulated wglChoosePixelFormatARB() function and enable multisampling of the entire scene.
osg::DisplaySettings actually works like a default value set of various display attributes. If there is no separate setting for a specific object, the default one will take effect; Otherwise the osg::DisplaySettings instance will not be put to use.
We are going to talk about the separate settings for creating graphics context and the osg::GraphicsContext class in Chapter 9

7.1.14 Stereo visualization

We have already experienced the charm of stereoscopic 3D films and photographs.
→James Cameron's Avatar
Anaglyph image is the earliest and most popular method of presenting stereo visualiyation.
others: NVIDIA's quad-buffering, horizontal or vertical split, horizontal or vertical interlace, ...

Fortunately, OSG supports most of these common stereo techniques, and can immediately relize one of them in the viewer with just a few commands:

```
osg::DisplaySettings::instance() -> setStereoMode( mode );
osg::DisplaySettings::instance() -> setStereo( true );
```

The method setSteroMode(0 selects a stereo mode from a set of enumerations, and the setStereo() meghod enables or disables it. Available stereo modes in OSG are: `ANAGLYPHIC, QUAD_BUFFER (NVIDIA ) , HORIZONTAL_SPLIT, VERTICAL_SPLIT` (DLP projector).
You may also use `LEFT_EYE or RIGHT_EYE` to indicate that the screen is used for left-eye or right-eye views.
for more stereo params, such as the eye separation, have a look at the API documentation.

7.1.15 Time for action - rendering naglyph stereo scenes

P183

7.1.16 wtf just happened?

in the `ANAGLYPHIC` mode, the final rendering result is always made yp of two color layers, with a small offset to produce a depth effect. Each eye of the glasses will see aslightly different picture, and their composition produces a sterograph image, which will be fused by our brain into a three dimensional scene.
OSG suports the anaglyphic stereo mode with a two-pass rendering scheme. The first pass renders the left eye image with a red channe color mast, and the second pass renders the right eye image with a cyan channel. the color mask is defined by the rendering attribute osg::ColorMask. It can be easily applied to state sets of nodes and drawables by using:

```
osg::ref_ptr <osg::ColorMask> colorMask = new osg::ColorMask;
colorMask -> setmask( true, true, true, true );
stateset -. setAttribute( colormask.get() );
```

stero mode often causes the scene graph to be rendered multiple times, which sloes down the frame rate as a side effect.

7.1.17 Rendering to textures

the render to textures technique allows developers to create textures based on a sub-scene's appearance in the rendered scene. These textures are then

"baked"into objects of coming scg via texture mapping. They can be used to create nice specital effects on the fly, or can be stored for subsequent deferred shading, multi-pass rendering, and other advanced rendering algorithms.
To implement texture baking dynamically, there are generally three steps to follow:

1. Create the texture for rendering.
2. Render the scene to the texture.
3. Use the texture as you want.

We have to create an empty texture object before putting it into use. OSG can create an empty osg::Texture object by specifying its size. The setTextureSIze() method defines the width and height of a 2D texture, and an additional depth parameter of a 3D texture.

The key to rendering a scene graph to the newly created texture is the attach() method of the osg::Camera class. This accepts the texture object as an argument, as well as a buffer component parameter, which indicates which part of the frame buffer will be rendered to the texture. For example, to attach the color buffer of a camera's sub-scene to the texture, we use:

```
camera -> attach ( osg::Camera::COLOR_BUFFER, texture.get() );
```

Other usable buffer components include the `DEPTH_BUFFER`, `STENCIL_BUFFER` and `COLOR_BUFFER0` to `COLOR_BUFFER15`(multiple render target outputs, depending on the gpu).
Continue setting suitable view and projection matrices of this camera, and a viewpoer to meet the texture size, and set the texture as an attribute of nodes or draeables. The texture will be updated with the camera's rendering result in every frame, dynamically carying with the alteration of the view matrix and the projection matrix.
Be aware that the main camera of a viewer is not suitable for attaching a texture. Otherwise there will be no ouputs to the actual window, which will make the screen pitch-dark. Of course, you ay ignore this if you are doing off-screen rendering and on't care of any visual effects.

7.1.18 Frame buffer, pixel buffer, and FBO

A concern is how to get the rendered frame bufdfer image into the texture object. A direct approach is to use the glReadpixels() method to return pixel data from the fram buffer, and apply the result to a glTexImage*() method. →easy to conceptualize and use, but will always copy data to the texture object, which is extremely slow.

The glCopyTexSubImage() method would be better in terms of improving the efficiency. However, we can still optimize the process. Rendering the scene directly to a target other than the frame buffer is a good idea. There are mainly two solutions for this:

1. the pixel buffer (pbuffer for short) extension can create an invisible rendering buffer with a pixel format descriptor, which is equivalent to a window. It should be destroyed after being used, as is done for the rendering window.

2. The frame buffer object (FBO for shor), which is sometimes better than pixel buffer in saving the storage space, can add application-created fram buffers and redirect the rendering output to it. It can either output to a texture object or a renderbuffer object, which is imply a data storage object.

OSG supports making use of different render target implementations: directly copying from the frame buffer pixel buffer, or FBO. It uses the method setRenderTargetImplementation() of the osg::Camera class to selsect a solution from them, for example:

```
camera -> setRenderTargetImplementation( osg::Camera::FRAME_BUFFER );
```

This indicates that the rendering result of Camera will be rendered to the attached texture by using the glCopyTexSubImage() method internally. In fact, this is the default setting of all camera nodes.
Other major implementations include `PIXEL_BUFFER` and `FRAME_BUFFER_OBJECT`

7.1.19 TFA - drawing aircrafts on a loaded terrain

In this section , we are going to integrate what we learned before to create a slightly complex example, which identifies all texture objects in a scene graph by using the osg::NodeVisitor utility, replaces them with a newly created shared texture, and binds the new texture to a render-to-textures camera. the texture is expected to represent more than a static image, so a customized simulation loop will be used to animate the sub-scene graph before calling the frame() method.

7.1.20 Summary

Chapter 8

Animating Scene Objects

OSG 's toolkits for real time animation:

- transformation animation
- key-frame animation
- skeletal animation
- and so on

8.1 Overview

- callback concepts and usage
- ease morions in different situations
- simple path animations
- complex key-frame and animarion channel systems
- generate character animations with skeleton system
- implement rendering state and texture animations

8.2 Taking references to functions

last chapter, animation via the sub-scene graph for dynamically rendering to textures.
→NON-RECOMMENDED: update the view matrix of the render-to-textures camera in the post-frame events →major issue: is in multithread context.

"post0frame" events may overlap with separated cull or draw threads, thus causing data access conflicts
To avoid those conflicts, we may consider employing a reference of these animating functionalities for the update traversal and let OSG decide the execution timeline and when to call these functionalities according to the reference. The reference passed to an executable code fragment is called a callback.

A callback triggered in the update traversal is called an **update callback**. There is also an **event callback** and a **cull callback** for executing in event and cull traversals, respectively. Instead of just using the address of functions as their references, OSG provides its own implementation of the execution operation, which is called functor. To customize the execution code, we have to override the callbacka functor's key operator or method, and attach it to a suitable scene object, for instance, a node or a drawable.

8.3 List of callbacks

There are several kinds of callbacks in the OSG scene graph and backend. Among them, the **osg::NodeCallback** class is an important implementer of update, event, and cull callbacks. It can be only attached to nodes. For srawables, we have **osg::Drawable::UpdateCallback, osg::Drawable::EventCallback** and **osg::Drawable::CullCallback** to achieve the same.

8.4 P195 switch nodes update traversal

what it says mate.

8.5 WTF just happened?

traverse(0 so far had two purposes:
→customizing nodes by overriding the traverse() method for own-exec code;
→callin gthe traverse(*) method of the osg::NodeVisitor class in order to continue the traversal while implementing node visitor Although these two oc urences have different params, they actually represent the same processing pipeline.

firstly, the traverse() method of node visitors, which has a single osg::Node param, simply calls the node's traverse() virtual method and passes itself as an argument.

Secondly, the node's traversing method must call its super calss's traverse() at the end of the implementation.it will then determine if there are child nodes to be traversed with the current visitor object (using the accept() method of child nodes).

Finally, the visitor in turn calls the apply() virtual method to receive various types of nodes as its argument, and realizes customized visiting behaviours thereafter. Since each apply() method must call the cisitor;s traverse() to end itself, the cycle comes back to the first step, until the whole scene graph is traversed. The entire diagram can be explained with following image:

P198

the callback's operator() method calls its traverse() in the third form, with a visitor and a node parameter. However, there is no need to worry about the complexity, as the only work it performs is to call the traverse() method of the visitor and continue the traversal. If you fail to call this in the callback's method, the traversal will simply be stopped and will return from current node at once.

→use addUpdateCallback() instead of setUpdateCallback() since it allows more Callbacks to be added later on.

8.6 Avoiding conflicting mods

osg can make the **draw traversal** which transfers data to the OpenGL pipeline run in a separated thread. It must be synched with other draw traversals in every frame, but part of the draw trav can usually overlap the **udate trav** coming grom the next grame, which improves the rendering efficiency and reduces frame latency.
→frame() method of osgViewer::Viewer will return while the drawing work is still active. Data changes in update callbacks could then conflict with the unfinished rendering procvess and cause unexpected behaviors, and even crashes.

solution: the setDataCariance() method, which belongs to the osg::Object class - the base class of all scene objects - This can be set to one of three enum values: UNSPECIFIED (default), STATIC, DYNAMIC.
→DUNAMIC obj in the sg must be processsed at the beginning of the **draw trav** That is, the rendering backend should ensure all nodes and scene objects that are specified as DYNAMIC have finished being drawn before starting the next frame's update and cull traversals. However, STATIC objects, which are considered to be unchanged during updating and drawindf , can thus be rendered later and won't hold back the fdrame rate.

By default, any newly-allocated objects are set to UNSPECIFIED, including nodes, drawables, state sets, and attributes. This allowsd OSG to predict the data variance. Reset like this:

```
node -> setDataVariance( osg::Object::DYNAMIC );
```

8.7 TFA P199 drawing a geometry dynamically

see p199 dear friend hihihi

8.8 wtfjh

remove setDataVariance() line →example still runs perfectly.
Tha's because UNSPECIFIED objedcts can decide if they will be dynamically modified in callbacks or not, and reset the data cariance to DYNAMIC automatically.

Change enum DUNAMIC to STATIC
→occasionally the renderinf result is flickering and there is an OpenGL error message "invalid operation" from the console. Those are acaused by thread conflicts.

Without calling dirtyDisplayList() method, OSG will ignore all dynamic drawable changes and make use of the **display list** commands storing the previous vertex and primitive data.
Also, without the dirtyBound() OSG will not know if the bounding box no longer firs the drawable size, and will make mistakes when doing view frustum culling work.

8.9 understanding wase motions

train from station A to B takes 15 minutes.
we want to simulate this by altering the transformation matrix of the train in an **update callback**. →put the train at station A at time point 0, and at station B at time point 15 (minutes)
→move it evenly in transition phase.
A heavily-employed method used here is the **linear interpolation**

→draws a straight line between two neighboring samples P0 and P1, returns the appropriate point P along the line, which can be used to represent translation and scale operations of nodes. It is commonly expressed in the followinf form:
$P = (1 - t) * P0 + t * P1$
→t is between 0 and 1
Unfortunately, the motion of a train is usually much more complex. It starts from station A, accelerates slowly, drives at an even speed, decelerates, and finally stops at station B. In that case lin interpol is shit.

→we have **ease motions** or ease functions. These are mathematical functions that are used to interpolate values between two endpoints. An ease motion usually generates non-linear results, in order to produce more natural effects.
The **osgAnimation** library defines a number of built-in ease motions. Each of them has at least two arguments: start value (usually 0) and a duration (usually 1) →produces results in the range [start value, start value + duration]
.
They can be applied to the start (lnMotion), to the end (OutMotion), or to both start and end of the animation (lnOut Motion). We will list them in the following table: P 204

To create a linear interpolation motion obj, we just type:

```
//Start value is 0.0, and duration time is 1.0
osg::ref_ptr <osgAnimation::LinearMotion> motion = new osgAnimation::LinearMoti
```

The examples/osganimationeasemotion file in the OSG sourcve code can help you to discover these e ease motions graphically.

8.10 Animatin the transformation nodes

path animations are the most commonly-used animations in graphics apps.
They can be used to describe a running car, a flight, a rotating ball, or the camera's motion. The path should always be set up first (position, rotation, scale values at different key time nodes)
When simulation loop is running, a transition state is calculated every frame, using the linear interpolation for pos and scale vectors, and **spherical linear interpolation** for the rotation quaternion. The slerp() method of osg::Quat is used internally here.

OSG provides the osg::AnimationPath class to encapsulate a time varying transformation path. It has an insert() method that can be used to insert a control point at a specific time. A control point, declared by the osg::AnimationPath::ControPoint class, accepts a pos value and optional rotation and scale values in order to construct the animation path. e.g.:

```
osg::ref_ptr<osg::AnimationPath> path = new osg::AnimationPath;
path -> insert ( t1, osg::AnimationPath::ControlPoint( pos1, rot1, scale1 ) );
path -> insert( t2, .. );
```

t1 and t2 are time nodes in seconds, and rot1 is an osg::Quat variable for representing the rotation of a object.

Besides that, we can set up the loop mode of the animation path with the setLoopMode() method. The default value is LOOP, that is, the animation will continuously run on the preset path over and over again.This param can be changed to `NO_LOOPING` (run once) or SWING (create a ping-pong path for other purposes.

After that, we attach the osg::AnimatrionPath object to a built-in osg::AnimationPathCallback object which is actually derived from osg::NodeCallback, and can help developers to control their animating scenes in an intuitive way.

8.11 TFA - making use of the animation path

P205

8.12 wtf happened?????

The osg::AnimationPath class uses a getMatrix() method to compute and return a transitional transformation matrix according to the two control points just before and after a given time. It is then applied to the host osg::MatrixTransform , osg::PositionAttitudeTransform , or osg::Camera node in order to make it move along the path. This is done by the osg::AnimationPathCallback class, which is actually an update callback for a specific purpose.

If the osg::AnimationPathCallback object is attached to any kind of nodes other than transformation nodes previously described, it will become invalid. It is also not suggested to use the animation path callback as event or cull callbacks, as this may lead to unexpected results.

8.13 have a go hero

An animation must be able to be stopped, reset, and fast-forwarded, which makes it easy to be controlled by users. The osg::AnimationPathCallback class provides the reset() , setPause() , setTimeMultiplier() , and setTimeOffset() methods to implement these common operations. For example, to restart a preset animation path, callback apcb at any time: apcb-¿setPause(false); apcb-¿reset(); In order to set the time offset to 4.0s and move forward through the animation at a 2x speed, just use: Apcb-¿setTimeOffset(4.0f); apcb-¿setTimeMultiplier(2.0f); Now, can you figure out how to create your own path animation player?

8.14 Changing rendering states

redering states can be animated too. A number of effects can be generated by altering the proprties of one or more rendering attributes, including fade-in and fade-out, density and variation of the atmosphere, fog, changing the direction of light beams, and so on. We can easily implement a state animation in the **update callback**. We may either tetrieve the attribute object from the arguments of the overridden method or just manage the object as a member variable of the user-defined callback. Remember to make use of **smart pointers** to ensure that the member attribute won't be automatically destroyed if it is no longer referenced.

The ease motion classes can be used to improve the animation quality. We must allocate an ease motion object with the start value and duration params and update it with a delta time as the time step size. For example:

```
osg::ref_ptr<osgAnimation::LinearMotion> motion = new osg::Animation::LinearMot
...
motion -> update( dt );
float value = motion -> getValue();
```

This creates a linear motion object with the X axis (time) ranging from 0.0 to 10.0.
The getValue() method uses specific formula on the current X value, and obtains a corresponding Y value.

You should add the osgAnimation library as a dependence if you would like the wase motion and more functionalities to be used in your projects.

8.15 TFA fading in

P209

Chapter 9

Interacting with Outside Elements

OSG has a GUI abstraction library.
→centralizes different windowing systems (MFC, Qt, GLUT, ...)
This chapter: How OSG interacts with other elements, for instance, input devices and windowing systems.

- How to handle keyboard, mouse, ... events with **customized event handers**
- create and handle user-defined events
- understand the intersection test of scene objects
- configure window traits, graphical context
- integrate rendered scene into windowing system

9.1 Various events

GUI is a type of interface object that allows computer users to interact with programs in many ways via **GUI events**.
widget element is defined to receive these user actions and transfer them to an event handler object.
Event handler is written by high-level devs to implement specific functionalities(pop up dialogs, dave current content of text editor.
problem? most gui frameworks not compatible with each other. →not convenient for direct use in OSG app.
→OSG provides basic interface for anyone qho wants to handle GUI Events:

osgGA::GUIEventHandler.
osgGA::GUIEventHandler should be attached to the sg with addEventHandler() and removed with removeEventHandler()
→a kind of callback that will automatically be called during the **event traversal** (Chapter05).

P232 When inheriting osgGA::GUIEventHandler to implement your own event handlers, the most important work is to override the handle() method. **it has two arguments** →osgGA::GUIEventAdapter parameter that supplies the received events
→osgGA::GUIActionAdapter parameter for feedback

9.2 Handling mouse and keyboard inputs

The osgGA::GUIEventAdapter class manages many osg supported events. The getEventType() method returns the current GUI event stored in an event adapter. Every time the overriding handle() method is called, we have to check this first to determine the event type and take appropriate countermeasures.

table with main event types in OSG and related methods used to get necessary event arguments.
→p233

There is another getModKeyMask() method (returns `MODKEY_CTRL`, `MODKEY_SHIFT`,... as bool).
Check if CTRL is pushed:

```
if ( ea.getModKeyMask()&osgGA::GUIEventAdapter::MODKEY_CTRL )
{
... // Related operations
}
```

9.3 TFA - Driving the cessna

see P234

9.4 Handling events within nodes

Node callbacks can directly perform node operations according to future user events, for example, moving the node when the user is pressing a key. Event handlers are more generic. They are useful in configuring global settings and handling events for all kinds of scene elements. In this example, event callback is easier to implement than event handlers, but remember, too many callbacks may also cause performance problems.

9.5 Adding customized events

OSG uses internal event queueto monage coming GUI events in a FIFO list. handle() method of each added event handler will be executed as many times as the size of the event queue.
The event queue class, named osgGA::EventQueue , allows new events to be pushed in with the addEvent() method at any time. Its argument is an osgGA::GUIEventAdapter pointer, which uses setting methods like setEvent-Type()and setButton() to define its behavior.

userEvent() method, which adapts user-defined events with a user data pointer as the argument. This user data can be used to represent any kind of customized event, for instance, the timer event described in the following section.

It is of no use to create a completely new event queue object. The viewer class has already defined one to operate on: viewer.getEventQueue()-¿userEvent(data); Here, the variable data is an object derived from osg::Referenced . After adding this new event, the event handler will receive a USER event and developers can then read from the getUserData() method of the handler and do anything they want.

9.6 TFA - creating a user timer

see p239

9.7 picking objects

picking functionality can allow user to click a button with mouse in rendered scene. shooting, closing door, ...
3 steps required:

1. event handler receives mouse events. e.g. mouse push comes with x and y positions of the cursor.

2. which part of sg is under cursor. **osgUtil** provides intersection tools. This includes set of intersections including the picked drawable, its parent node path, intersecting point and so on.

3. intersection result helps picking objects or making them fight.

9.8 INTERSECTION

uses node visitor mechanism to reduce time.
more efficient than OpenGL's selection feature.
→osgUtil::INtersectionVisitor (dervied from osg::NodeVisitor)
→tests nodes' bounding volumes against the input intersector

The osgUtil::IntersectionVisitor object takes an osgUtil::Intersector derived object as the argument of its constructor.

→see page 244 for the coordinate system table

Chapter 10

saving and loading files

- file I/O mechanism implemented in OSG
- list of supported file formats
- OSG pseudo loaders
- cusomize OSG plugin interface / user - defined formats
- class wrappers for seriaized I/O of OSG native formats

10.1 understanding file I/O plugins

I/O naming convention: .osg file format example: under Windows: `osgdb_osg.dll` Under Linux it's `osgdb_osg.so` Both with prefix `osgdb_` and the following name represents the file extension
required data files must exist in specified relative or absolute paths or in the OSG serach path defined by the environment variable `OSG_FILE_PATH`.

10.2 discovery of specified extension

seaching and location a plugin for the handling of a specified file type, 2 steps:

1. plugin list in osgDB::Registry class.
   designed as singleton. instance() method to obtain.
   chain-of-responsibility design: each plugin (called reader-writer in OSG) tries to process extension of input file, and pass it off to the next plugin in the list if the extension is unrecognizable to that plugin.

2. if all reader-writers fail, OSG will use the extension as a keyword to find plugin from external shared module.
   that is the `osgdb_<ext>` library file. `<ext>` represents extension string example: p 265
   define extension string with FileExtensionAlias() method.

10.3 supported file formats

huge list on p.266 - 270.

10.4 the pseudo loader

extension makred as **pseudo-loader** means they are not file extensions, but add asuffix to the end of the real filename to indicate the file should be read by the specified plugin example on p270.
some pseudo loaders need/have parameters. example p270.

10.5 tfa reading files from the Internet

10.6 writing your own plugins

extending the virtual reader-writer interface, OSG allows custom file formats.
→the virtual interface is defined by the osgDB::ReaderWriter class.
→it has several important virtual methods to be used or re-implemented to achieve reading and writing functionalities.
see page 276.

for the implementation of the readNode() method read p277 example.

readNode() returns: osgDB::ReaderWriter::ReadResult, which isn't intuitive, doesn't return node pointer.
You essentially test the result and then return the node if correct.

Chapter 11

developing visual components

- create geometries as billboards
- text in scene (2D/3D)
- design particle system = animation
- cast shadows on scene objects
- theory and implementation of special effects

11.1 creating billboards in a scene

billboard + 2D image always facing a designated direction.
used for many special effects (explosions, flares, sky, clouds, trees)
→any object can be treated as a billboards with itself cached as the texture, while looking from a distance.
→very popular
→osg::BillBoard class. Derived from osg::Geode. All it's children (osg::Drawable objects) face designated direction (viewer's viewpoint, ...).
important method: setMode(). Used to determine the rotaion behavior, with following enums: `POINT_ROT_EYE, OPINT_ROT_WORLD, AXIAL_ROT` (explanation p.292)
Drawables in the osg::BillBoard node should have a pivot point position, which is specified via the overloaded addDrawable() method:

```
billboard->addDrawable( child, osg::Vec3(1.0f, 0.0f, 0.0f) );
```

Drawables also need unified initial front face orientation, used for computing rotation values. Initial orientation isi set by setNormal(). Each added drawable must ensure that its front face orientation is int he same firection as this normal value; otherwise the billboard results may be incorrect.

11.2 TFA creating banners facing you

11.3 creating texts

important for vr. Stats on screen, labeling 3D objects, logging, debugging..
texts have a font, size, alignment, layout (left-to-right, right-to-left, ...), resolution,...
OpenGL doesn't support this, but OSG dows.
osgText library implements font and text. Requires `osgdb_freetype` plugin
→plugin can load and parse true type fonts (freetype dependancy).
→returns an osgText::Font instance.
entire process: can be described with osgText::readFontFile() function.
osgText::Textbase class is the pure base class of all OSG text types.
Derived from osg::Drawable.
no support for display lists by default.
sublclass osgText::Text is used to manage glat characters in the world coord.
important methods: setFont(), setPosition(), setCharacterSize(), setText();

11.4 TFA - writing cessna description

11.5 creating 3D texts

OSG also provides support for 3D texts in the scene graph. Each character will be extruded with a depth parameter and finally rendered with OpenGL's vertex array mechanism. The implementer class, osgText::Text3D , is also derived form osgText::Textbase and thus has nearly the same methods as osgText::Text . It requires an osgText::Font3D instance as the font parameter, which can be obtained by the osgText::readFont3DFile() function.

11.6 tfa - 3D texts in world space

Both 2D and 3D texts can be transformed by their parent nodes.
setPosition() method of osgText::TextBase only sets the location under the relative reference frame of the text object's parent.
same thing to setRotation() and setAxisAlignment().
exception: SCREEN alignment mode:

```
text->setAxisAlignment( osgText::TextBase::SCREEN );
```

mimics the billboard technique of scene objects, and makes the text (either osg::Text or osg::Text3D) always face the viewer.
placing landmarks on earth or cities as billboards is a very common operation, and can be implemented with the SCREEN mode. In this case, rotation and parent transformations are not available and should not be used, as they may cause confusion and potential problems.

11.7 particle animations

smoke, dust, explosions, fluid, fire, rain. more difficult to build and manage a complete particle system rather than cconstruct other simple scene objects.
→**osgParticle class**
→complex particle systems, most of which may be extended and overridden using inheritance, if user-definded algorithmsr are needed.

osgParticle::Particle represents the atomic particle unit.
→often used as design template before the simularion loop atarts, copied and regenerated by the particle system in run-time to render massicve particles.

osgParticle::ParticleSystem , manages the creation, updating, rendering, and destruction of all particles. It is derived from osg::Drawable , so it can accept different rendering attributes and modes, just like normal drawables. It should be added to an osg::Geode nod, as the last class.

osgParticle::Emitter abstract class, defines number and propterties of particles generated every frame.
has 3 sub-controllers:

1. osgParticle::Placer sets the initial Position of every particle
2. osgPartivle::Shooter sets the initioal velocities
3. osgPartivle::Counter determines how many particles should be created

osgParticle::program manips the position, velocity and other prroperties of each individual particle suring its lifetime.
osgParticle::ModularProgram descendant class form osgParticle::Program, is composed of a list of osgParticle::Operator subclasses to perform operations on existing particles.

particles need to be updated and re computed with osgParticle::ParticleSystemUpdater. This should be done as the last act.

Chapter 12

Improving Rendering Efficiency

12.1 Learn goals

1. multithreaded operations and rendering in OSG
2. concept of scene culling, occlusion culling technique
3. ways to improver rendering performance, by modifying and sharing geometries and textures.
4. dynamic paging mechanism and its utilization in handling huge datasets

12.2 OpenThread Basics

OpenThreads: lisghtweight, cross-platform thread API.
For OSG classes and appsl
OpenThreads::Thread = thread object.
OpenThreads::Mutex = mutex for locking data that may be shared by different threads.
→mutual exclusion object. allows multiple program threads to share the same resource (file access) but not simultaneously.
OpenThreads::Barrier = barrier object. can be used where we want a set of threads to wait for each other.
OpenThreads::Condition = like barrier, used for thread synchronization.

how to create a thread:
derive the OpenThreads::Thread base class.
re-implement some of its virtual methods

convenient global functions:

- GetNumberOfProcessors() available proccessors

- SetprocessorAffinityOfCurrentTHread() sets the processor affinity of the current thread. **affinity**: which processor uses which thread. should be called when the rthread is currently running.

- CurrentThread() static method of OpenThreads::Thread. Returns a pointer to the current running thread instance.
- YieldCurrentThread(0 static method of OpenThreads::Thread. Yields current thread and lets other threads take over the control of the processor.
- microSleep() static method of OpenThreads::Thread. make current thread sleep for a specified number of microseconds. It can be used in single-threaded apps too.

12.3 TFA: using a separate data receiver thread