OpenGL – Beginners Guide

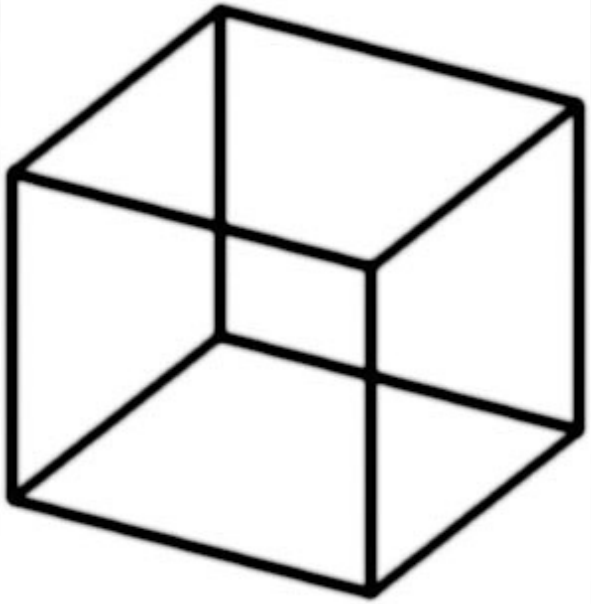
From Basics to Intermediate

Introduction

3D Graphics Overview

- What actually is 3D Graphics ?

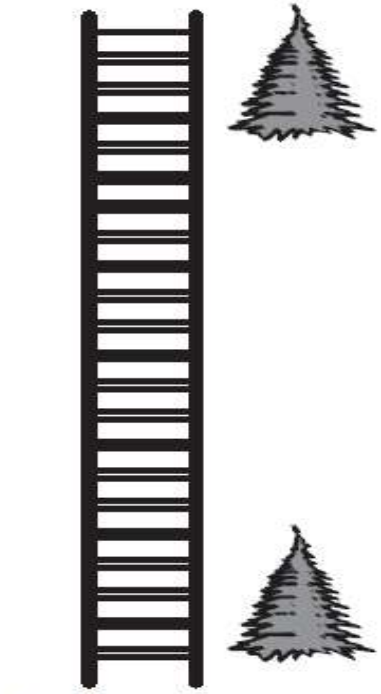
3D computer graphics are actually two-dimensional images on a flat computer screen that provide an illusion of depth, or a third dimension



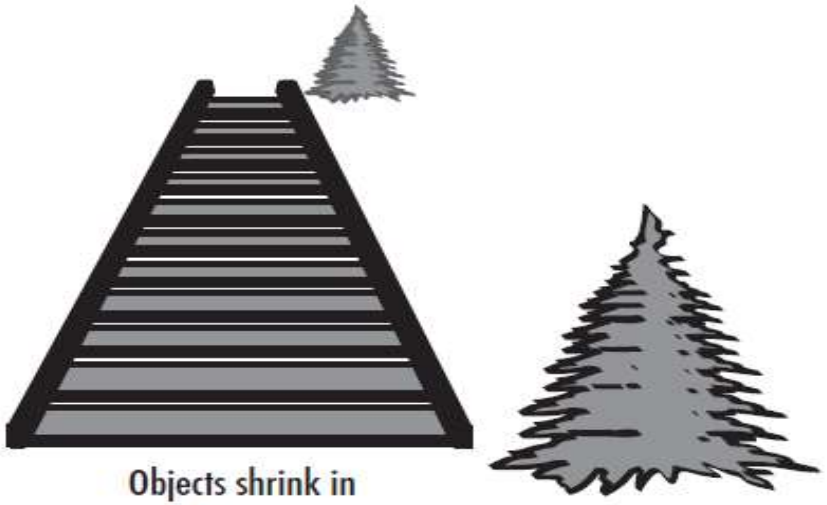
3D Graphics Overview

- Perspective

Perspective refers to the angles between lines that lend the illusion of three dimensions



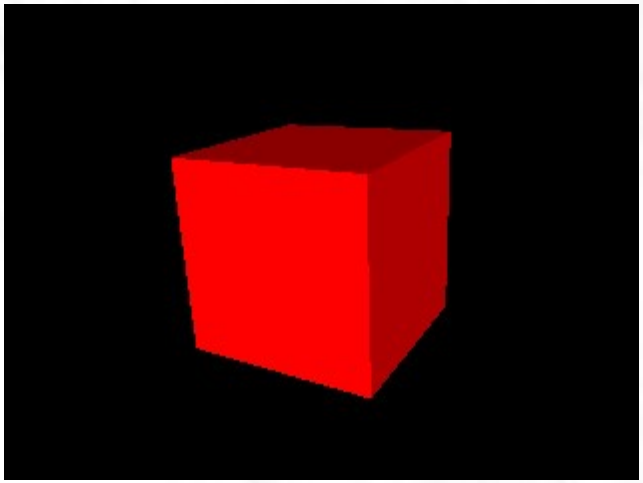
Everything same size

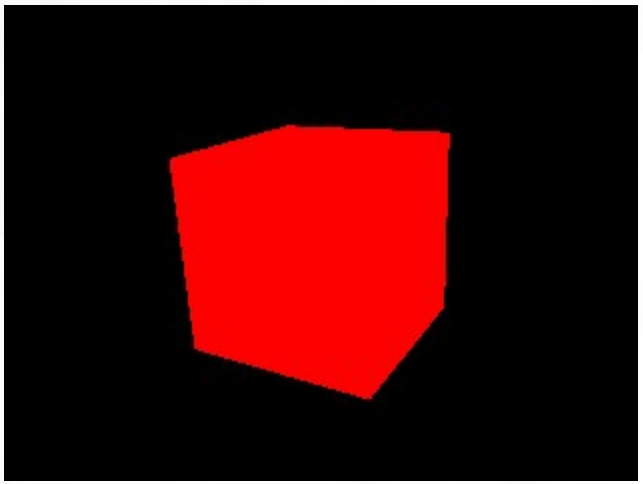


Objects shrink in distance

3D Graphics Overview

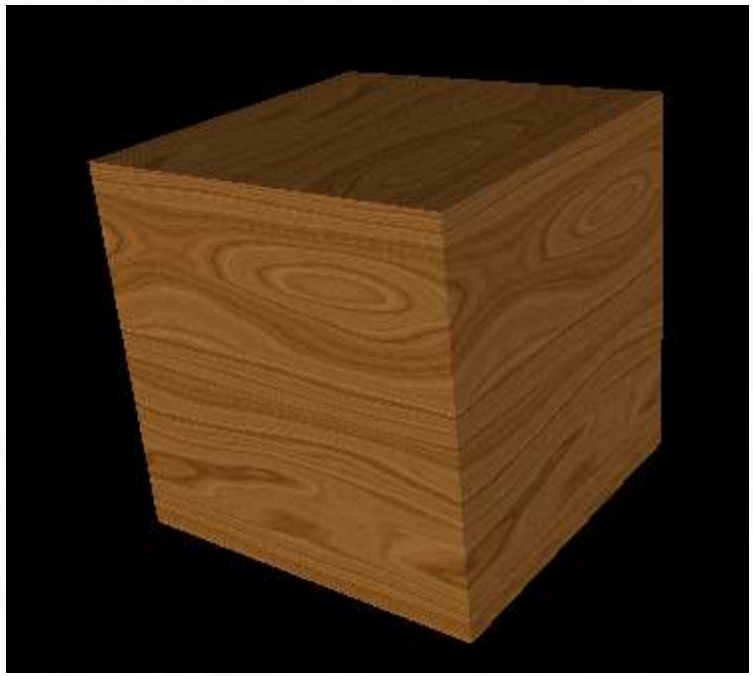
- Light and Shading





3D Graphics Overview

- Texture



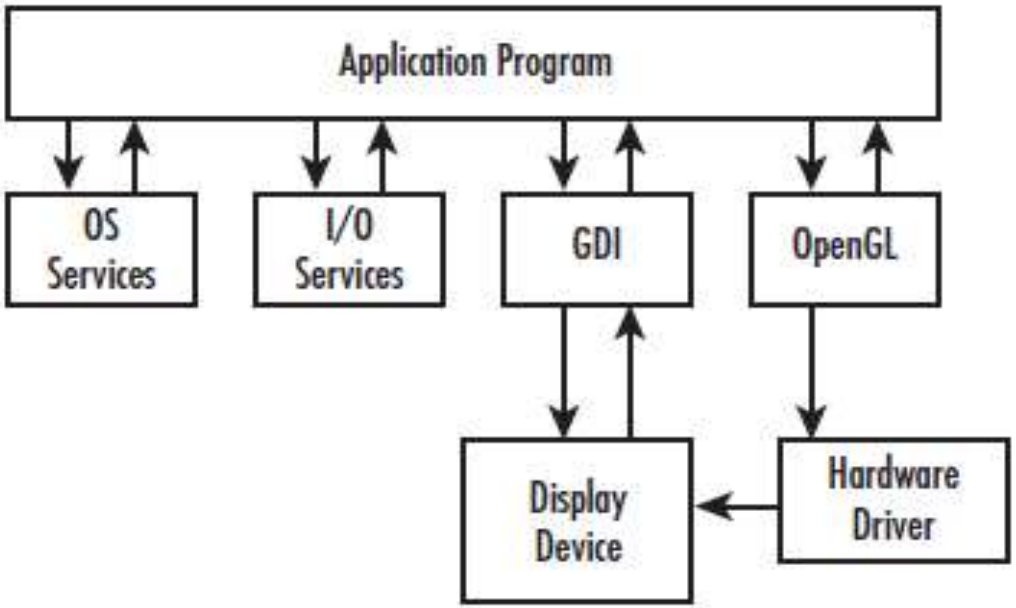
Getting Started

What is OpenGL ?

- OpenGL is strictly defined as “a software interface to graphics hardware.” In essence, it is a 3D graphics and modeling library that is highly portable and very fast.
- An API, Not a Language,It is more like the C runtime library
- Initially developed by SGI as IRIS GL, later became an open standard and known as OpenGL

How Does OpenGL Work?

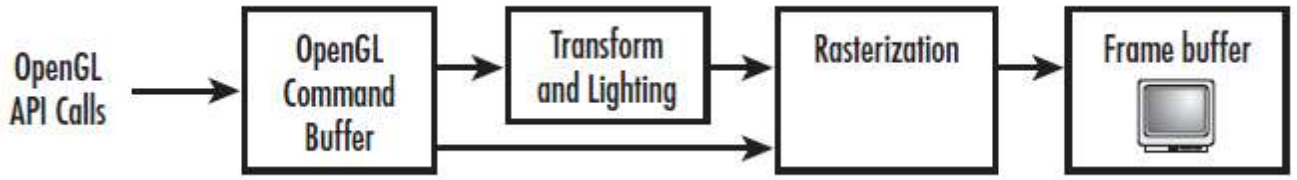
- The Structure of OpenGL Program looks mostly like this



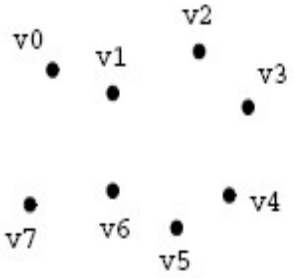
How Does OpenGL Work?

- OpenGL Pipeline

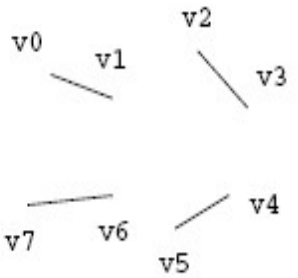
The word pipeline is used to describe process that can take two or more distinct stages or steps.The figure shows a simplified version of the OpenGL pipeline.



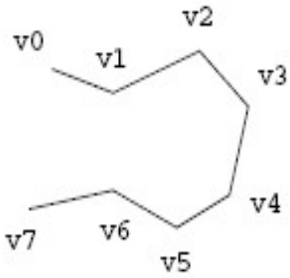
OpenGL Primitives



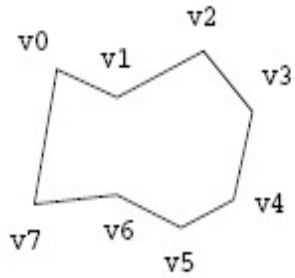
GL_POINTS



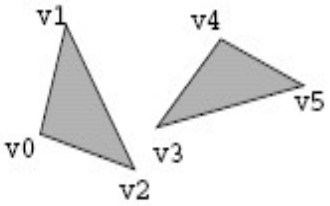
GL_LINES



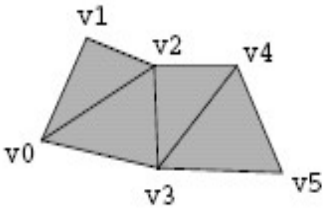
GL_LINE_STRIP



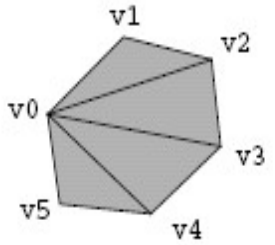
GL_LINE_LOOP



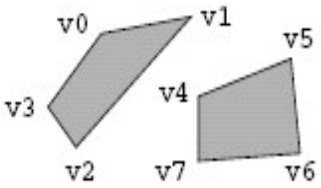
GL_TRIANGLES



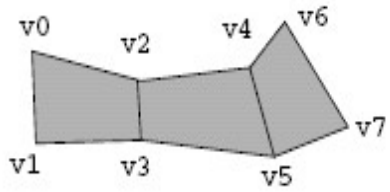
GL_TRIANGLE_STRIP



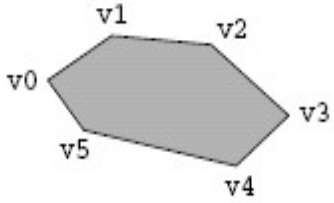
GL_TRIANGLE_FAN



GL_QUADS



GL_QUAD_STRIP



GL_POLYGON

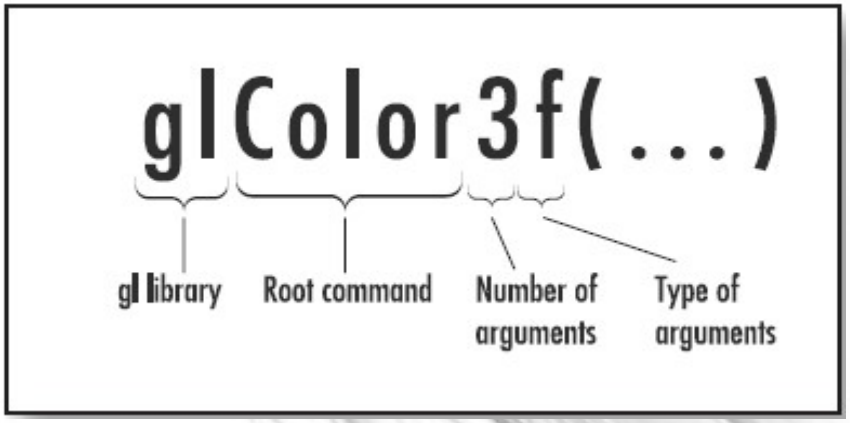
API Specifics

- Header files

```
#include<windows.h>
#include<gl/gl.h>
#include<gl/glu.h>
```

- Function-Naming Conventions

<Library prefix><Root command><Optional argument count><Optional argument type>



OpenGL Extensions

- OpenGL extensions are a means for OpenGL implementations to provide new or expanded functionality that the core of OpenGL does not provide

```
#include "glew.h"
```

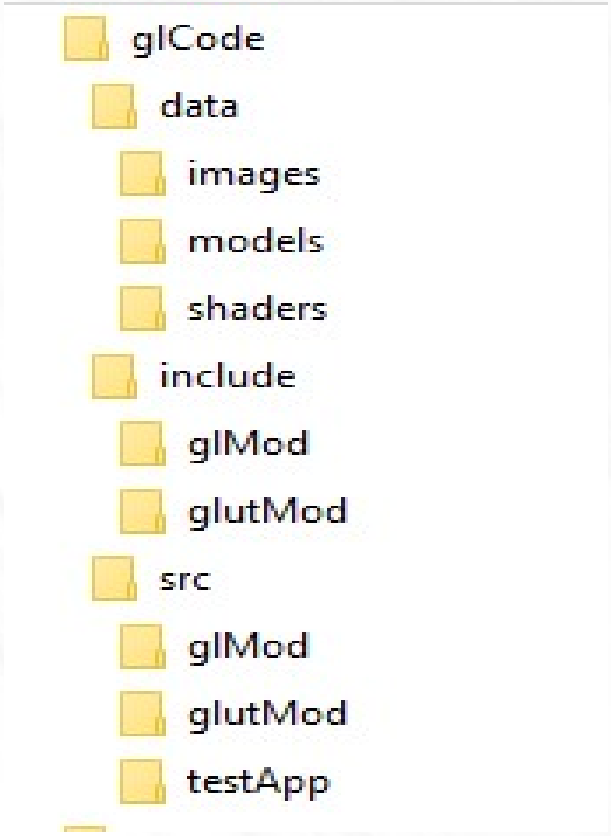
- Extension-Naming Conventions
  - GL_EXT_blend_color
  - glBlendColorExt
  - GL_ARB_blend_color
  - glBlendColorArb

- Usage

```
const char *extensions = glGetString(GL_EXTENSIONS);

if ( strstr(extensions, "WGL_EXT_swap_control" != NULL) ){

   wglSwapIntervalEXT =(PFNWGLSWAPINTERVALEXTPROC)wglGetProcAddress("wglSwapIntervalEXT");

   if( wglSwapIntervalEXT != NULL )

        WglSwapIntervalEXT(1);

}
```

Source code distribution

- Source file structure



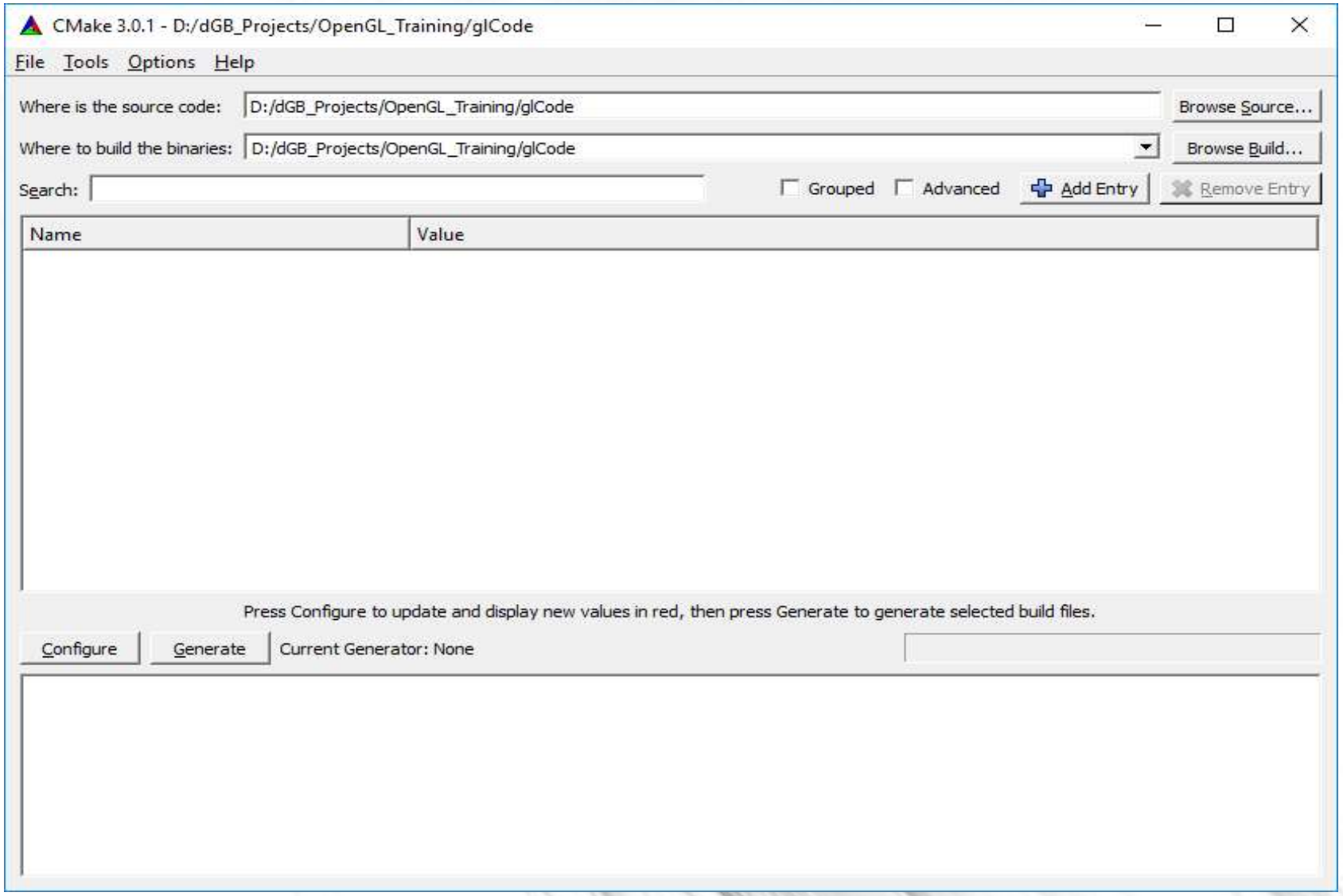
A small talk on CMake

- What is CMake

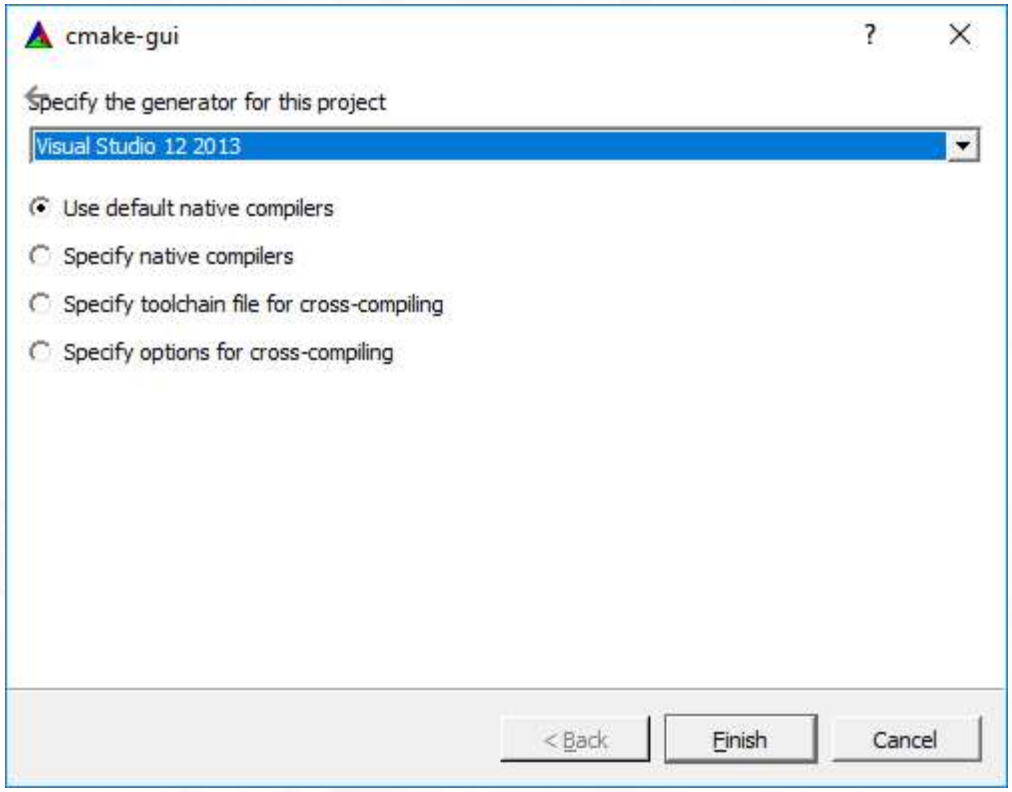
CMake is an open-source, cross-platform family of tools designed to build, test and package software. CMake is used to control the software compilation process using simple platform and compiler independent configuration files, and generate native makefiles and workspaces that can be used in the compiler environment of your choice. The suite of CMake tools were created by Kitware in response to the need for a powerful, cross-platform build environment for open-source projects such as ITK and VTK.

https://cmake.org/

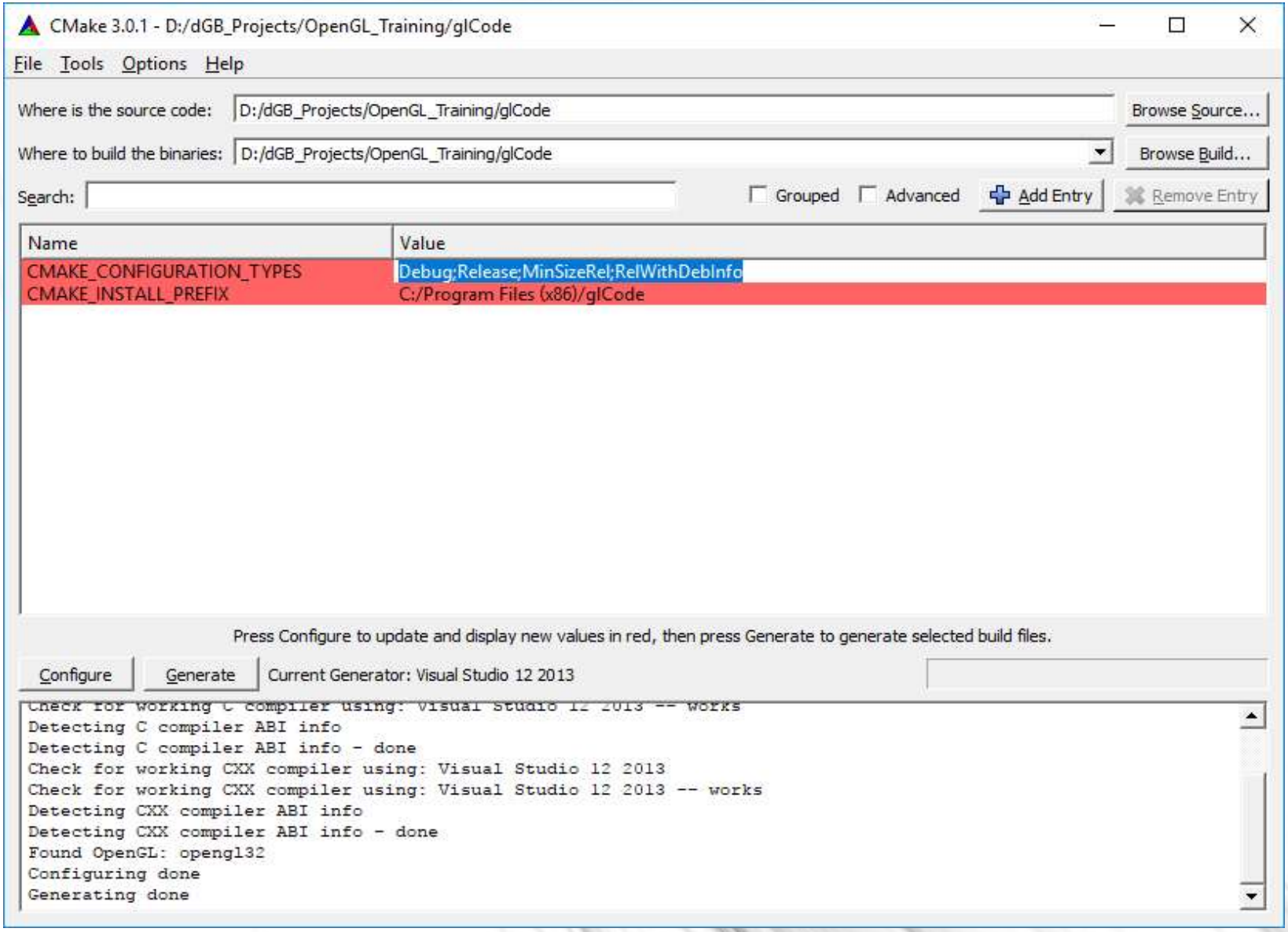
A small talk on CMake



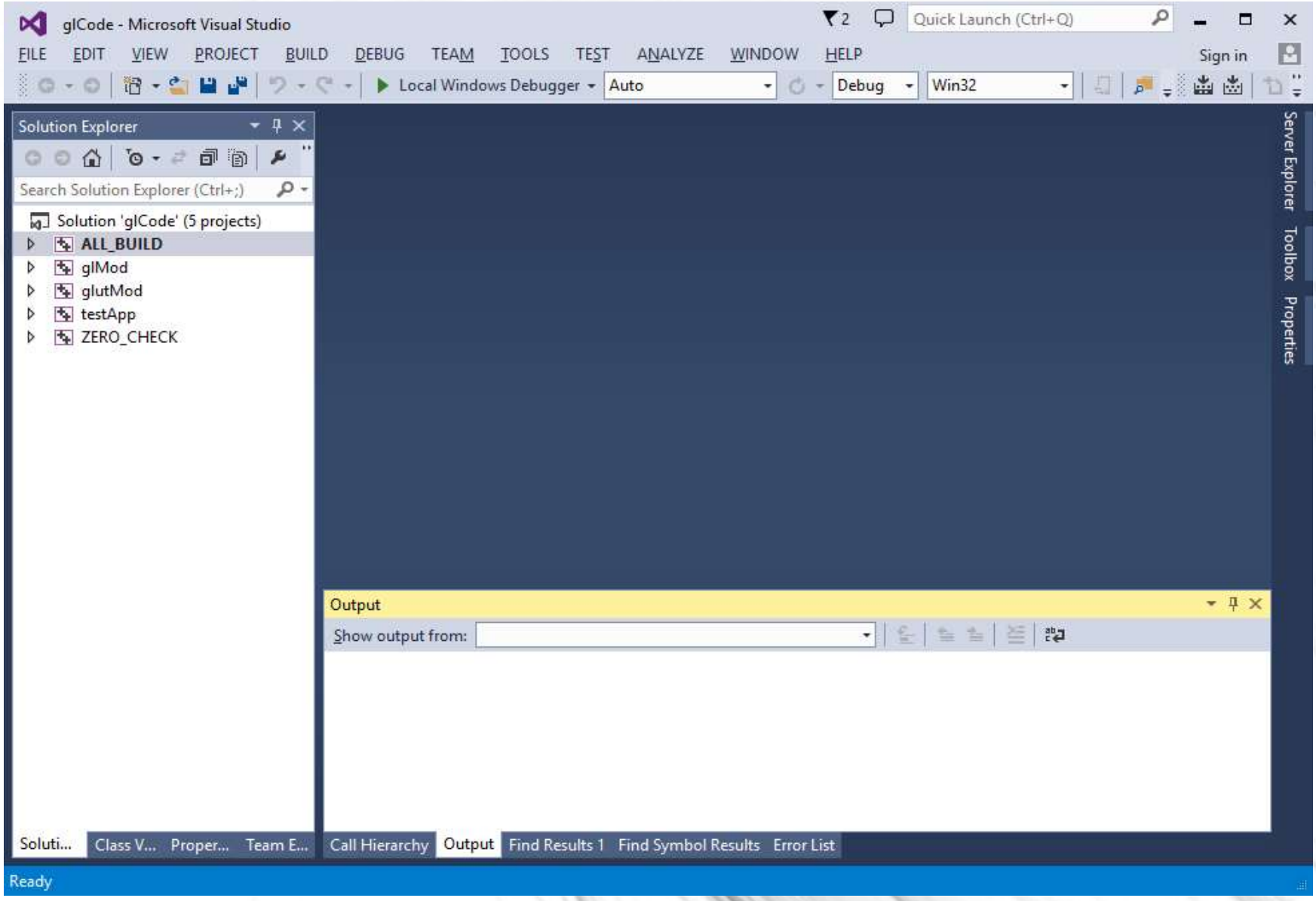
A small talk on CMake



A small talk on CMake



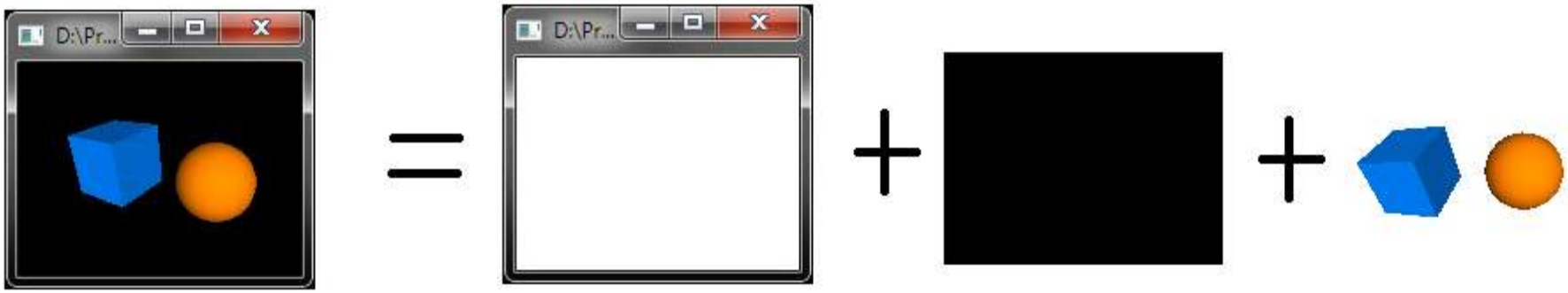
A small talk on CMake



GLUT – What and How

- What is GLUT

GLUT (pronounced like the glut in gluttony) is the OpenGL Utility Toolkit, a window system independent toolkit for writing OpenGL programs. It implements a simple windowing application programming interface (API) for OpenGL. GLUT makes it considerably easier to learn about and explore OpenGL programming. GLUT provides a portable API so you can write a single OpenGL program that works across all PC and workstation OS platforms.



Drawing Shapes

OpenGL State Machine

- OpenGL is a state machine. You put it into various states (or modes) that then remain in effect until you change them

```
glEnable( GL_LIGHTING );

glDisable( GL_LIGHTING );
```

- Saving and Restoring States

Values are correspondingly restored with this command:

```
void glPushAttrib(GLbitfield mask);


void glPopAttrib(GLbitfield mask);


glPushAttrib( GL_TEXTURE_BIT | GL_LIGHTING_BIT );
```

Drawing Points, Lines, Triangles, Quads etc.

- Drawing Examples (Let's do hands on)

```
glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
glDisable( GL_LIGHTING );
 glPointSize( 5.0 );
glBegin( GL_POINTS );
    glVertex3f( x1, y1, z1 );
    glVertex3f( x2, y2, z2 );
  glEnd();
```

Drawing simple 3D Shapes

- Drawing Cube (With Code Example)
- Drawing Sphere (With Code Example)

Drawing complex 3D Shapes

- Drawing Complex Shapes (With Code Example)

Loading 3D Models

- Loading .obj file (With Code Example)

Color, Material, Light and Normal

Color, Material, Light and Normal

- Color function basics

```
glColor3ub( 255, 0, 0 );
glColor3f( 1.0, 0, 0 );
```

Color, Material, Light and Normal

- Concept of Material

Matrerial determines how light will be reflected from the surface

```
float diffuse_reflectivity[] = { 0.0, 0.25f, 0.25f, 1.0f };
float ambient_reflectivity[] = { 1.0, 1.0f, 1.0f, 1.0f };
glMaterialfv( GL_FRONT, GL_AMBIENT, ambient_reflectivity );
glMaterialfv( GL_FRONT, GL_DIFFUSE, diffuse_reflectivity );
```

Color, Material, Light and Normal

- Light – Ambient, Diffuse, Specular

Ambient light.

Ambient light means the light that is already present in a scene, before any additional lighting is added. It usually refers to natural light, either outdoors or coming through windows etc.

Diffuse light

The diffuse part of an OpenGL light is the directional component that appears to come from a particular direction and is reflected off a surface with an intensity proportional to the angle at which the light rays strike the surface

Specular light

Like diffuse light, specular light is a highly directional property, but it interacts more sharply with the surface and in a particular direction. A highly specular light (really amaterial property in the real world) tends to cause a bright spot on the surface it shineson, which is called the specular highlight

Color, Material, Light and Normal

- Blending

How the source and destination colors are combined when blending is enabled is controlled by the blending equation. By default, the blending equation looks like this

$$Cf = (Cs * S) + (Cd * D)$$

Here, Cf is the final computed color, Cs is the source color, Cd is the destination color, and S and D are the source and destination blending factors. These blending factors are set with the following function:

```
glBlendFunc(GLenum S, GLenum D);
```

Color, Material, Light and Normal

- RGBA, Transparency - Alpha blending

```
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
```

This function tells OpenGL to take the source (incoming) color and multiply the color (the RGB values) by the alpha value. Add this to the result of multiplying the destination color by one minus the alpha value from the source. Say, for example, that you have the color Red (1.0f, 0.0f, 0.0f, 0.0f) already in the color buffer. This is the destination color, or Cd. If something is drawn over this with the color blue and an alpha of 0.6 (0.0f, 0.0f, 1.0f, 0.6f), you would compute the final color as shown here:

Cd = destination color = (1.0f, 0.0f, 0.0f, 0.0f)

Cs = source color = (0.0f, 0.0f, 1.0f, 0.6f)

S = source alpha = 0.6

D = one minus source alpha = 1.0 – 0.6 = 0.4

Now, the equation

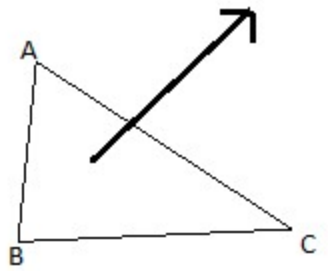
Cf = (Cs * S) + (Cd * D)

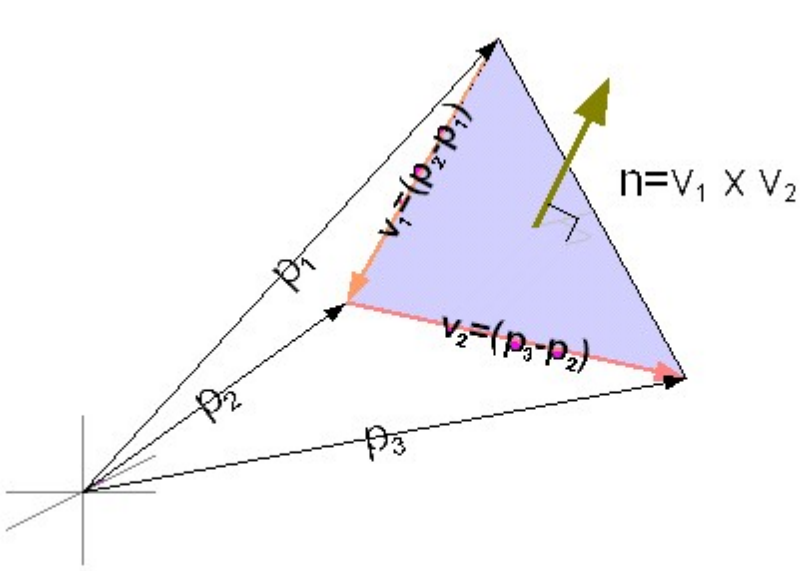
evaluates to

Cf = (Blue * 0.6) + (Red * 0.4)

Color, Material, Light and Normal

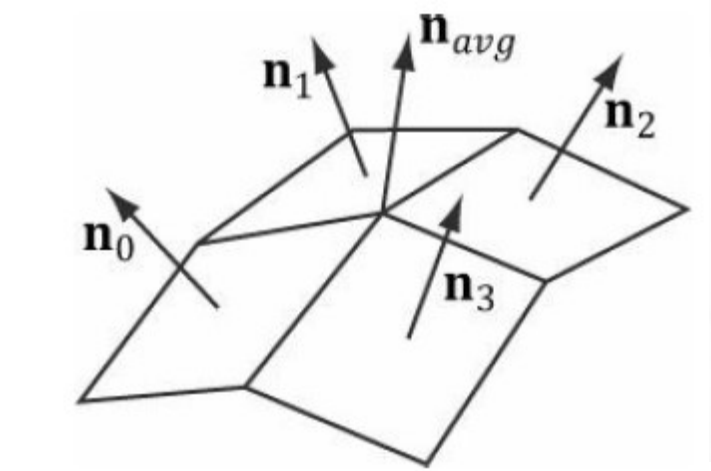
- Surface Normals

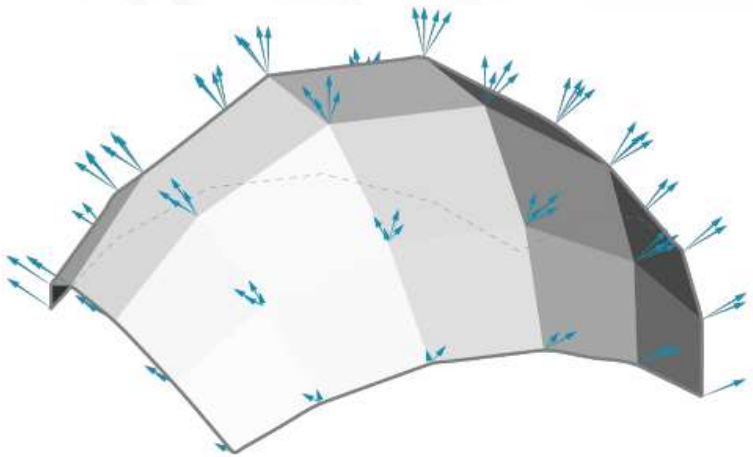




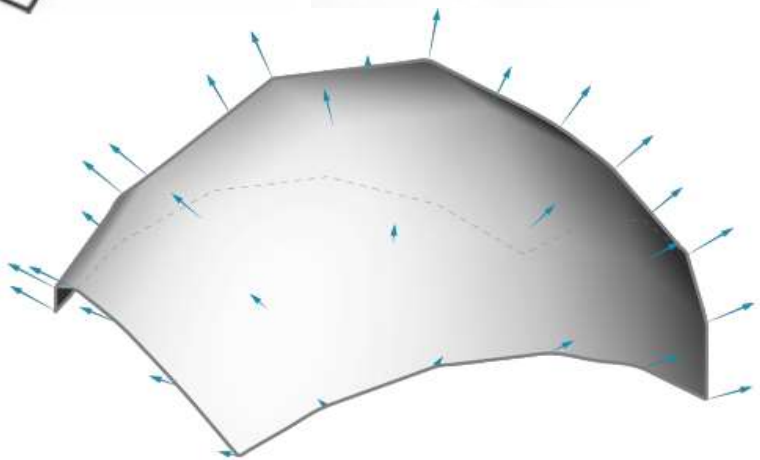
Color, Material, Light and Normal

- Vertex Normals





1

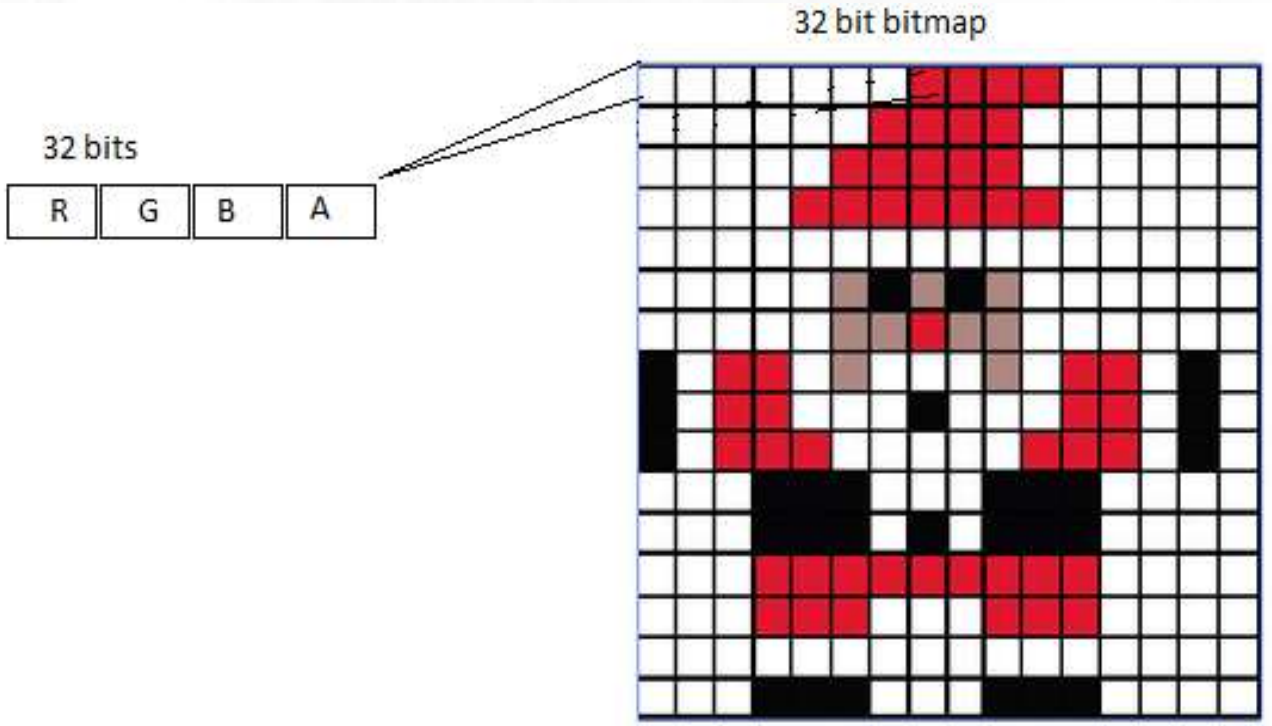


2

Texture Mapping

Texture Mapping

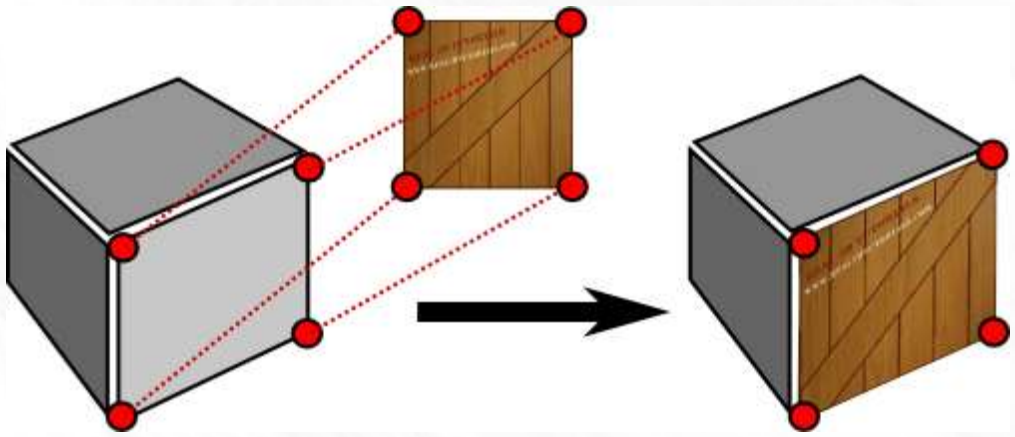
- Bitmap images



Texture Mapping

- Texture mapping

The application of patterns or images to three-dimensional graphics to enhance the realism of their surfaces.

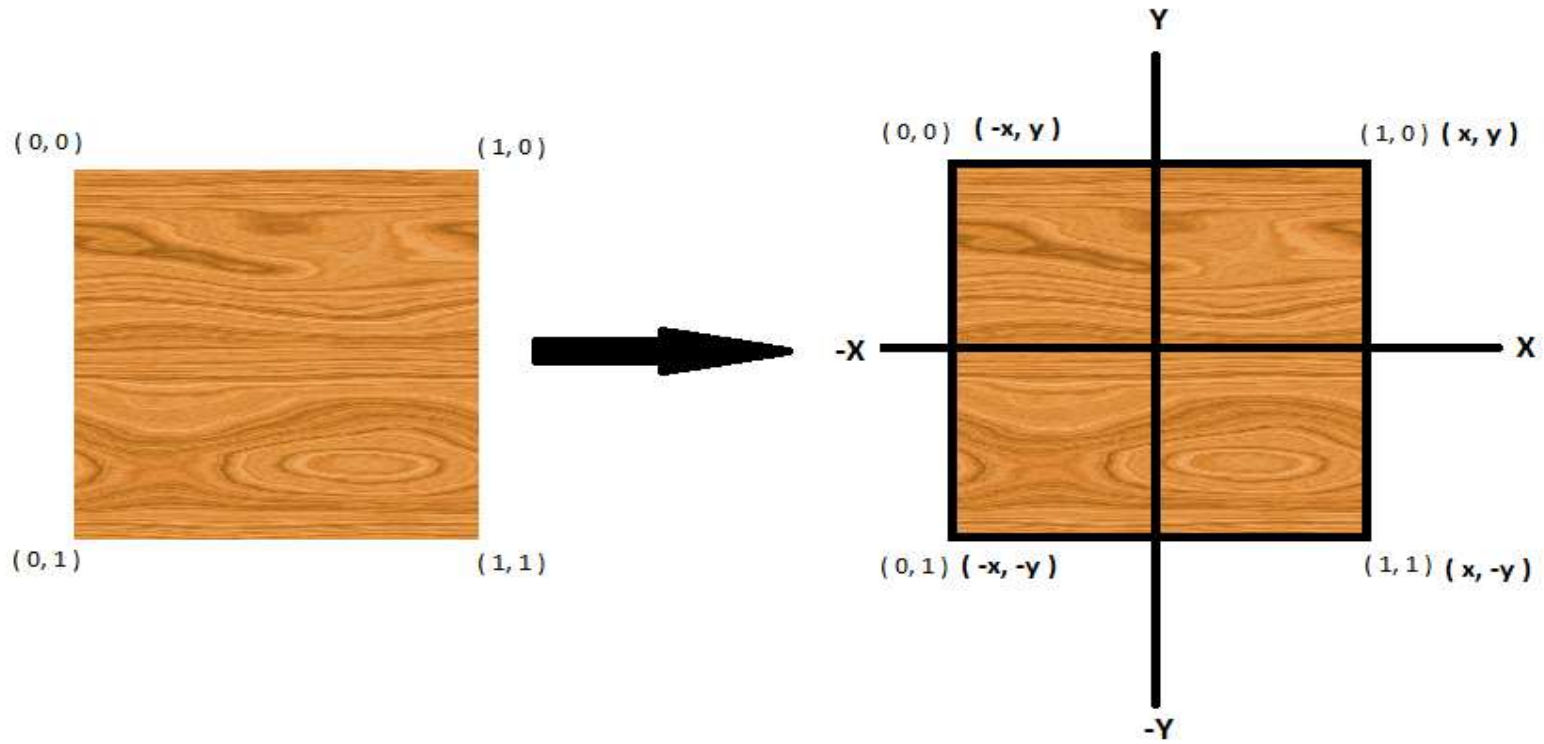


Texture Mapping

- Texture Coordinates

Texture coordinates are specified as floating-point values that are in the range 0.0 to 1.0.

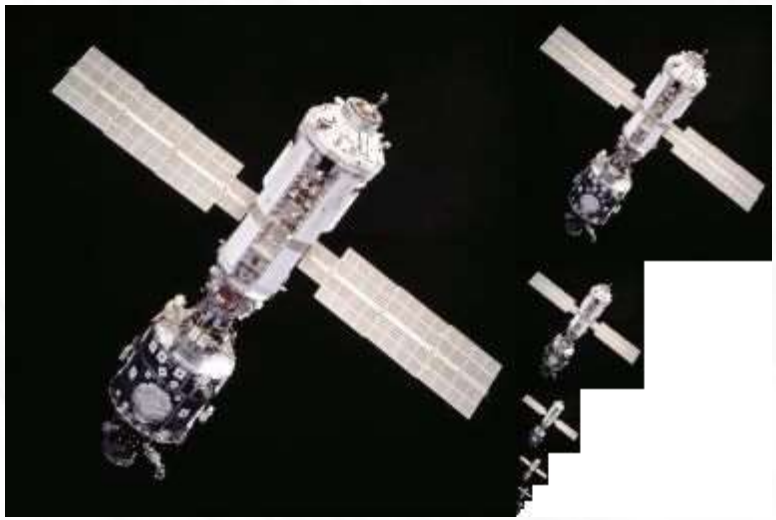
Texturecoordinates are named **s, t, r**, and q (similar to vertex coordinates x, y, z,and w), supporting from one- to three-dimensional texture coordinates, and optionally a way to scale the coordinates.The application of patterns or images to three-dimensional graphics to enhance the realism of their surfaces.



Texture Mapping

- Mipmaps

In computer graphics, mipmaps (also MIP maps) or pyramids [1][2][3] are pre-calculated, optimized sequences of images, each of which is a progressively lower resolution representation of the same image. The height and width of each image, or level, in the mipmap is a power of two smaller than the previous level.



Texture Mapping

- Texture Environment

The process by which the final fragment color value is derived is called the texture environment function (glTexEnv()) Several methods exist for computing the final color, each capable of producing a particular effect. One of the most commonly used is the modulate function. For all practical purposes the modulate function multiplies or modulates the original fragment color with the texel color.

```
glTexEnvi( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE );

glTexEnvi( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE );

glTexEnvi( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_BLEND );
```

Texture Mapping

- Texture Objects / Multiple Textures

Texture objects are an important new feature in release 1.1 of OpenGL. A texture object stores texture data and makes it readily available. You can now control many textures and go back to textures that have been previously loaded into your texture resources. Using texture objects is usually the fastest way to apply textures, resulting in big performance gains, because it is almost always much faster to bind (reuse) an existing texture object than it is to reload a texture image using glTexImage*D().

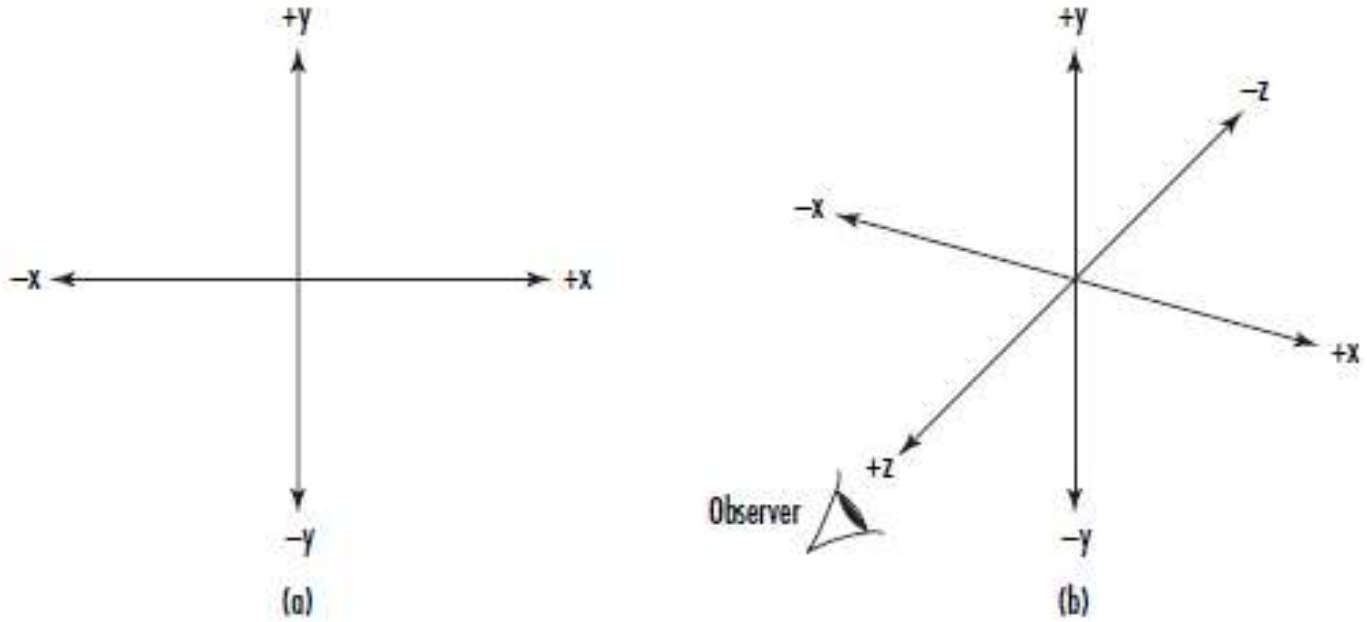
```
glGenTextures(GLsizei n, GLuint *textures);

glBindTexture(GLenum target, GLuint texture);
```

Transformations Matrices et al.

Transformations Matrices et al.

- Eye Coordinates (Viewing Transformations)

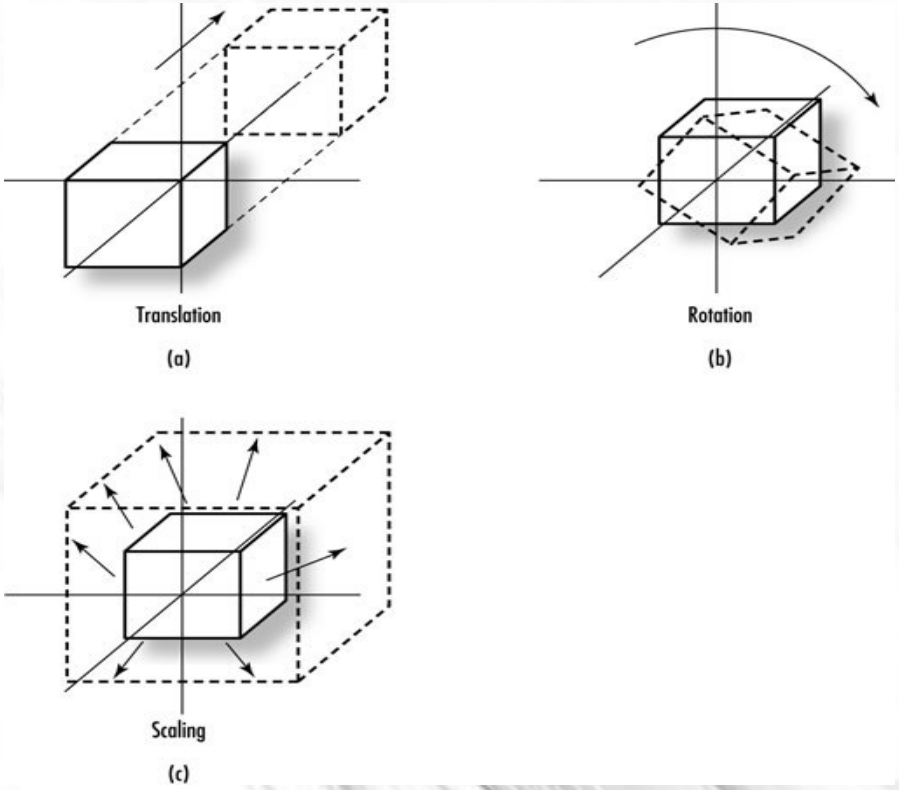
The viewing transformation is the first to be applied to your scene. It is used to determine the vantage point of the scene. By default, the point of observation in a perspective projection is at the origin (0,0,0) looking down the negative z-axis (“into” the monitor screen). This point of observation is moved relative to the eye coordinate system to provide a specific vantage point. When the point of observation is located at the origin, as in a perspective projection, objects drawn with positive z values are behind the observer.In an orthographic projection, however, the viewer is assumed to be infinitely far away on the positive Z axis, and can see everything within the viewing volume.



Transformations Matrices et al.

- Modeling Transformation

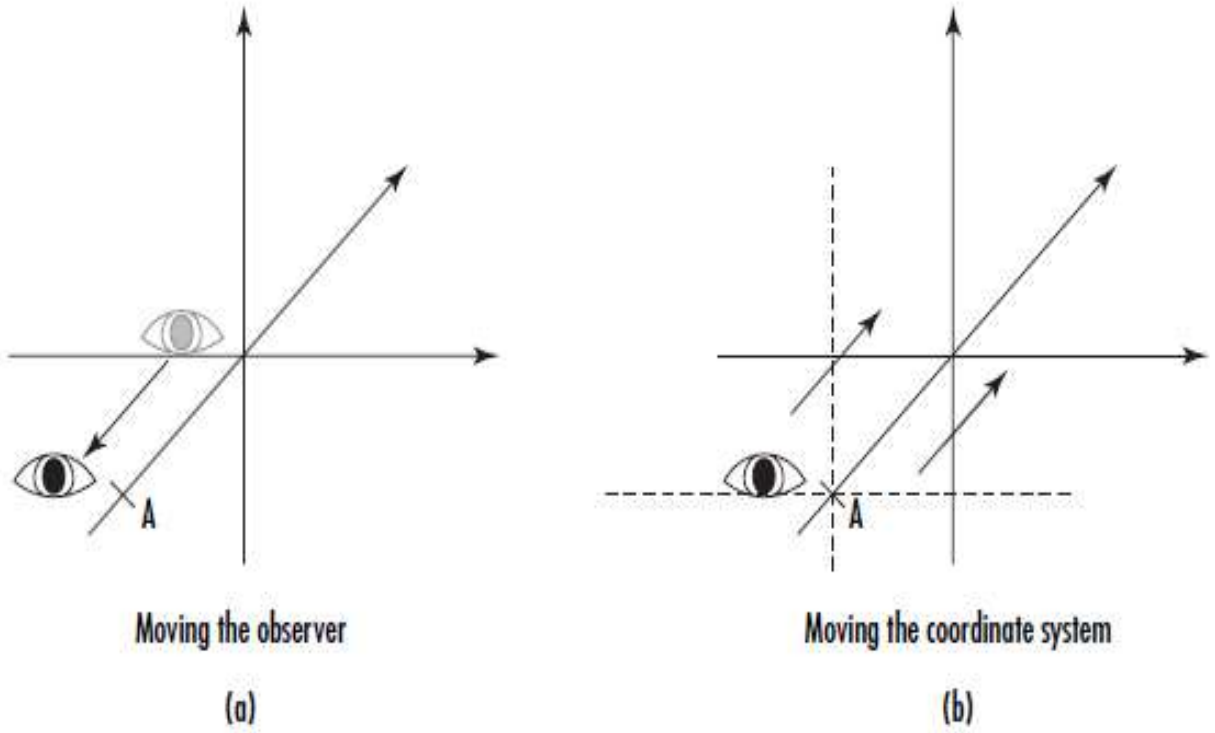
Modeling transformations are used to manipulate your model and the particular objects within it. These transformations move objects into place, rotate them, and scale them



Transformations Matrices et al.

- Modelview Duality

The viewing and modeling transformations are, in fact, the same in terms of their internal effects as well as their effects on the final appearance of the scene. The distinction between the two is made purely as a convenience for the programmer. There is no real difference visually between moving an object backward and moving the reference system forward

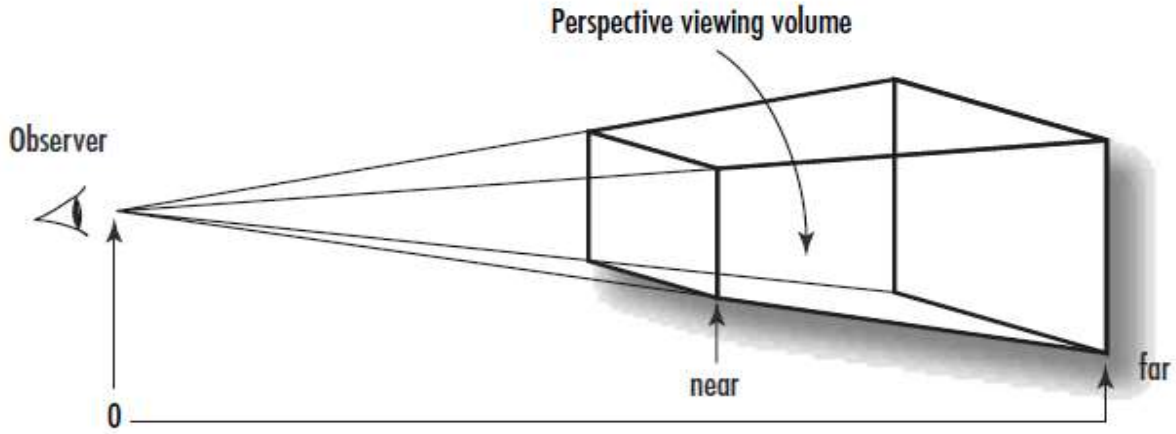


Transformations Matrices et al.

- Projection Transformations

The projection transformation is applied to your vertices after the modelview transformation.This projection actually defines the viewing volume and establishes clipping planes.The clipping planes are plane equations in 3D space that OpenGL uses to determine whether geometry can be seen by the viewer

- ***Orthographic:*** In an orthographic, or parallel, projection, all the polygons are drawn onscreen with exactly the relative dimensions specified. Lines and polygons are mapped directly to the 2D screen using parallel lines, which means no matter how far away something is, it is still drawn the same size, just flattened against the screen. This type of projection is typically used for rendering two-dimensional images such as blueprints or two-dimensional graphics such as text or onscreen menus.
- ***Perspective*****:** A perspective projection shows scenes more as they appear in real life instead of as a blueprint. The trademark of perspective projections is foreshortening, which makes distant objects appear smaller than nearby objects of the same size

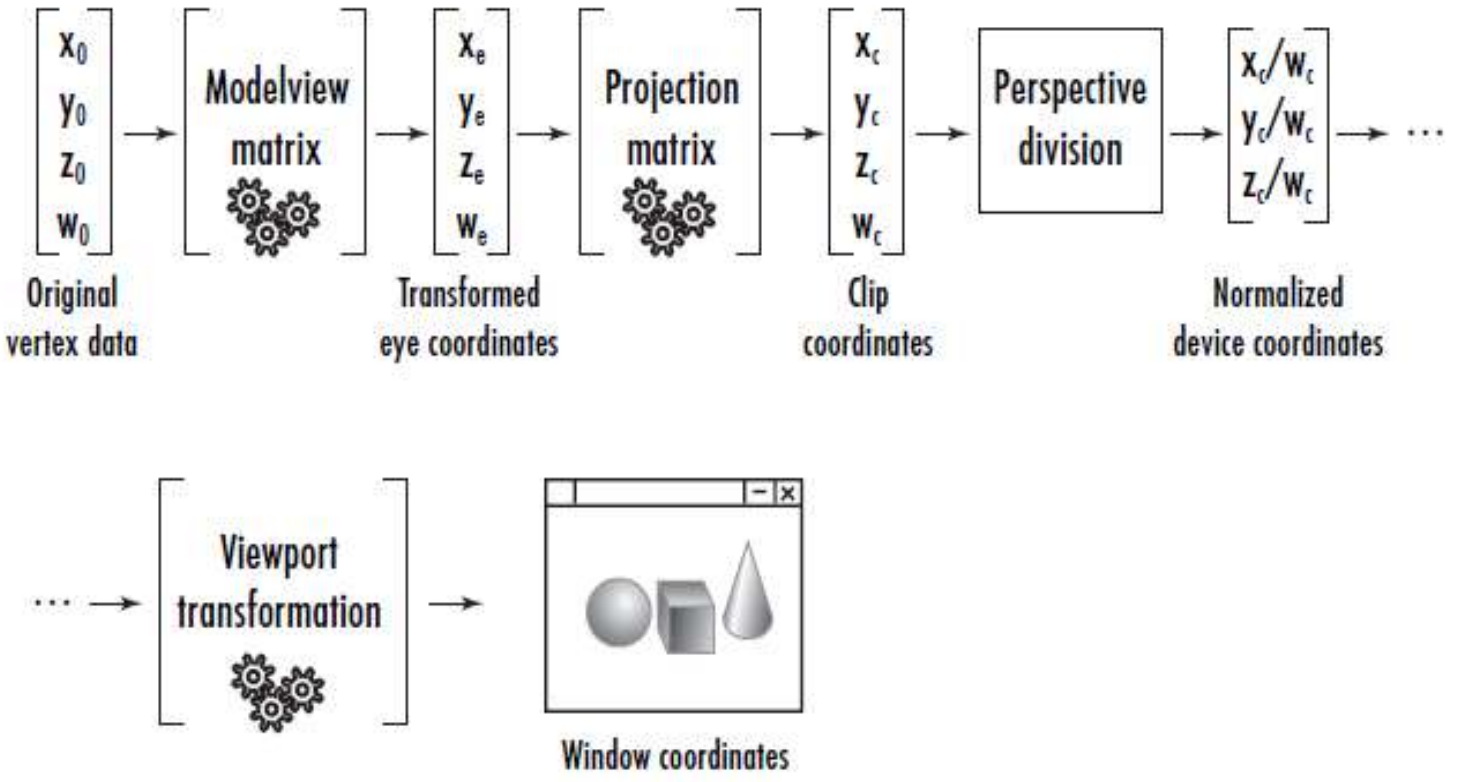


Transformations Matrices et al.

- Viewport Transformations

When all is said and done, you end up with a two-dimensional projection of your scene that will be mapped to a window somewhere on your screen. This mapping to physical window coordinates is the last transformation that is done, and it is called the viewport transformation

- Transformation Pipeline



Transformations Matrices et al.

- Basic Transformations => Translation, Scaling, Rotation
  - Translation

    glTranslatef(GLfloat x, GLfloat y, GLfloat z);
  - Scaling

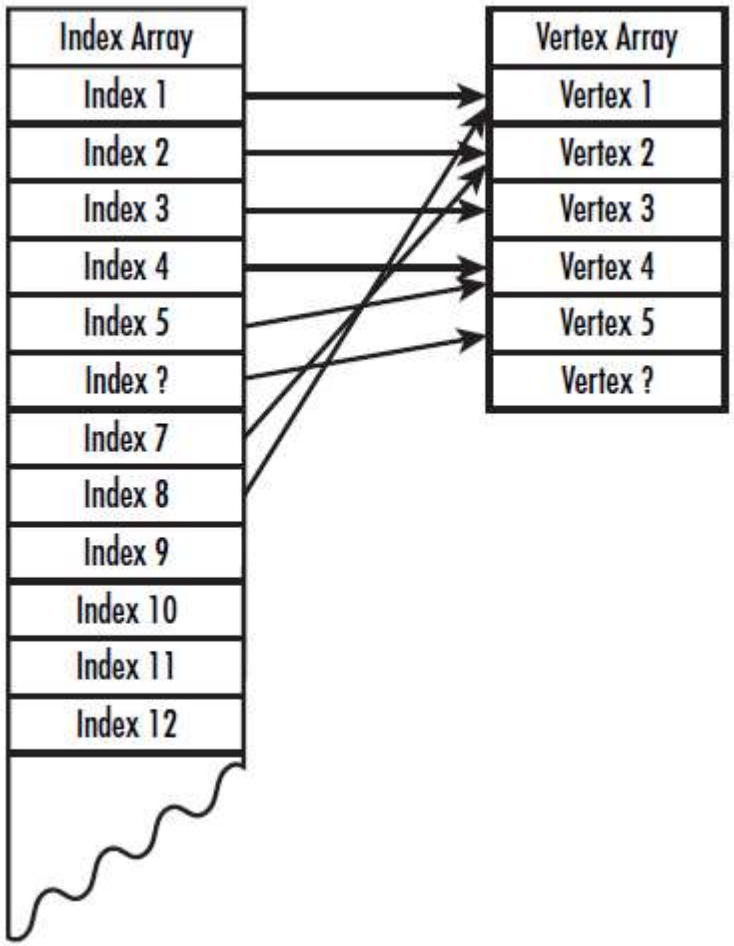
    glScalef(GLfloat x, GLfloat y, GLfloat z);
  - Rotation

    glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z);

Optimization Techniques

Optimization Techniques

- Using Arrays and Indices



Optimization Techniques

- Revisit OpenGL Extensions

Will be using some extensions to use advanced techniques like VBO

```
#include "Gl/glew.h"
```

Optimization Techniques

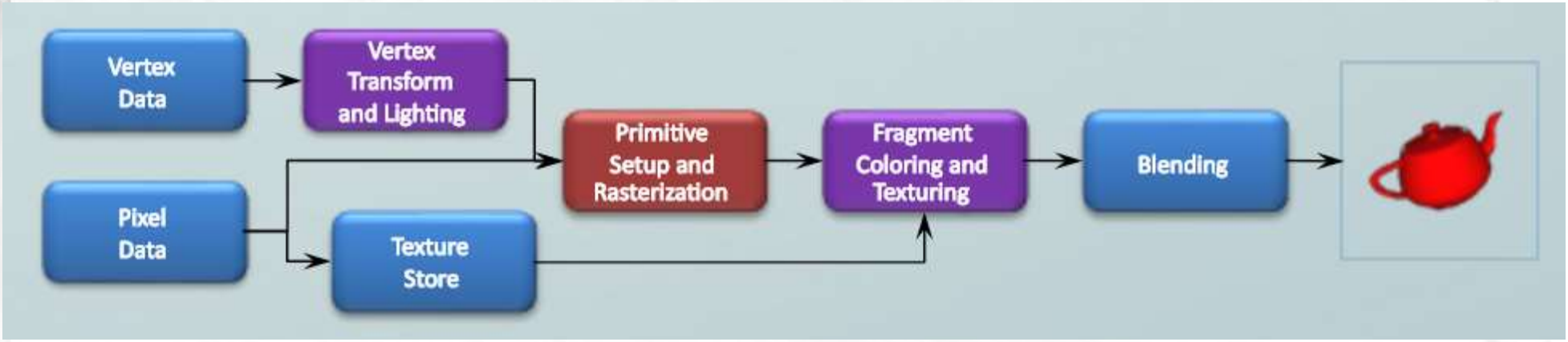
- Using Vertex Buffer Objects (With code examples)

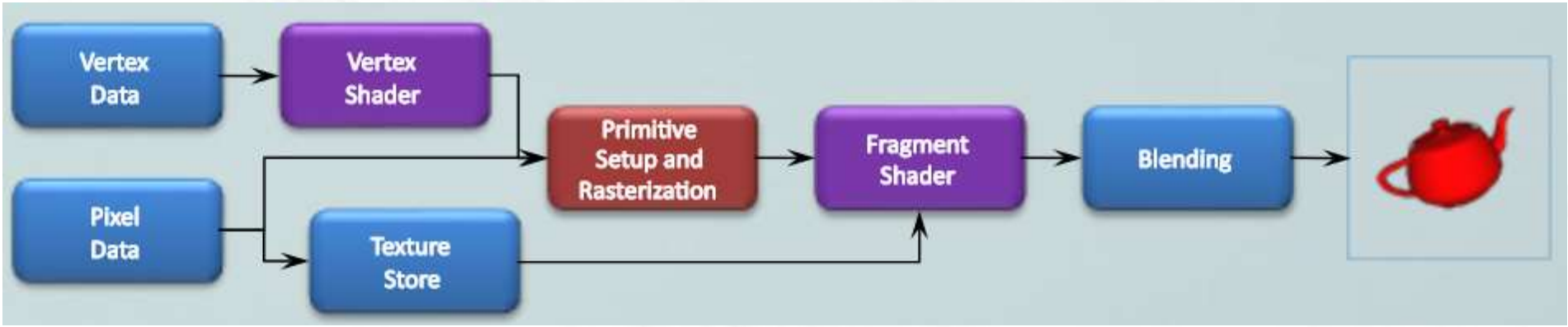
A Vertex Buffer Object (VBO) is an OpenGL feature that provides methods for uploading vertex data (position, normal vector, color, etc.) to the video device for non-immediate-mode rendering. VBOs offer substantial performance gains over immediate mode rendering primarily because the data resides in the video device memory rather than the system memory and so it can be rendered directly by the video device

Advanced OpenGL – GLSL/Shaders

Advanced OpenGL

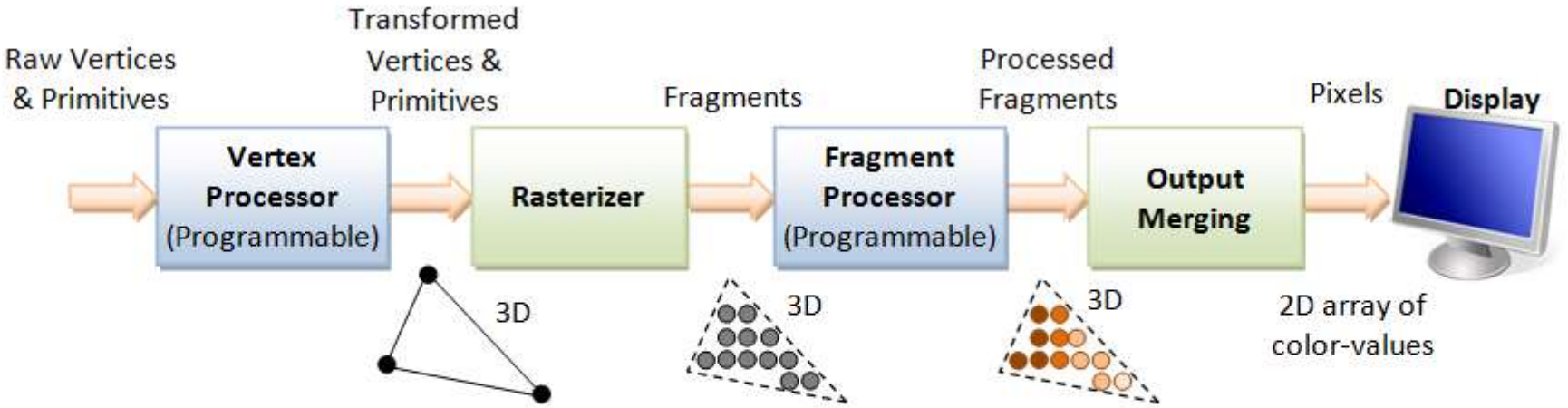
- The Programmable Pipeline - Shaders





Advanced OpenGL

- The Programmable Pipeline - Shaders



3D Graphics Rendering Pipeline: Output of one stage is fed as input of the next stage. A vertex has attributes such as (x, y, z) position, color (RGB or RGBA), vertex-normal (n_x, n_y, n_z), and texture. A primitive is made up of one or more vertices. The rasterizer raster-scans each primitive to produce a set of grid-aligned fragments, by interpolating the vertices.

Advanced OpenGL

- OpenGL Shading Langauge – GLSL

Attributes − These variables hold the input values of the vertex shader program. Attributes point to the vertex buffer objects that contains per-vertex data. Each time the vertex shader is invoked, the attributes point to VBO of different vertices.

Uniforms − These variables hold the input data that is common for both vertex and fragment shaders, such as light position, texture coordinates, and color.

Varyings − These variables are used to pass the data from the vertex shader to the fragment shader.

Advanced OpenGL

- Vertex Shading – Simple Specular Lighting

Advanced OpenGL

- Fragment Shading – Toon Shading

Advanced OpenGL

- Realistic Visual Effects Shading – Earth with Atmosphere

Acknowledgments

- OpenGL Super Bible Fourth Edition
- Wikipedia (www.wikipedia.org)
- dGB Earth Sciences (www.dgbes.com)

Thank You