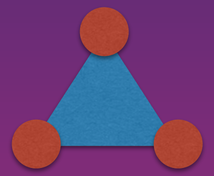
Shaders

Part 1.

The rendering pipeline.



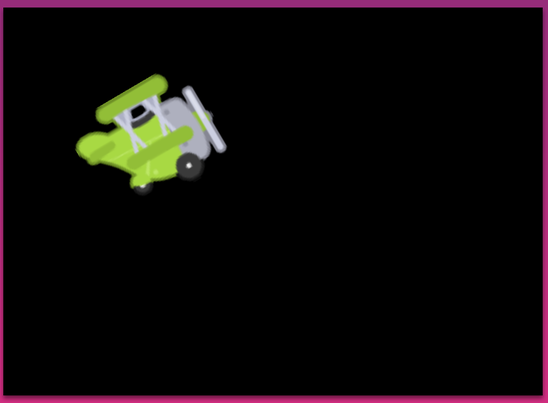
MODELVIEW MATRIX

PROJECTION MATRIX

VERTEX SHADER

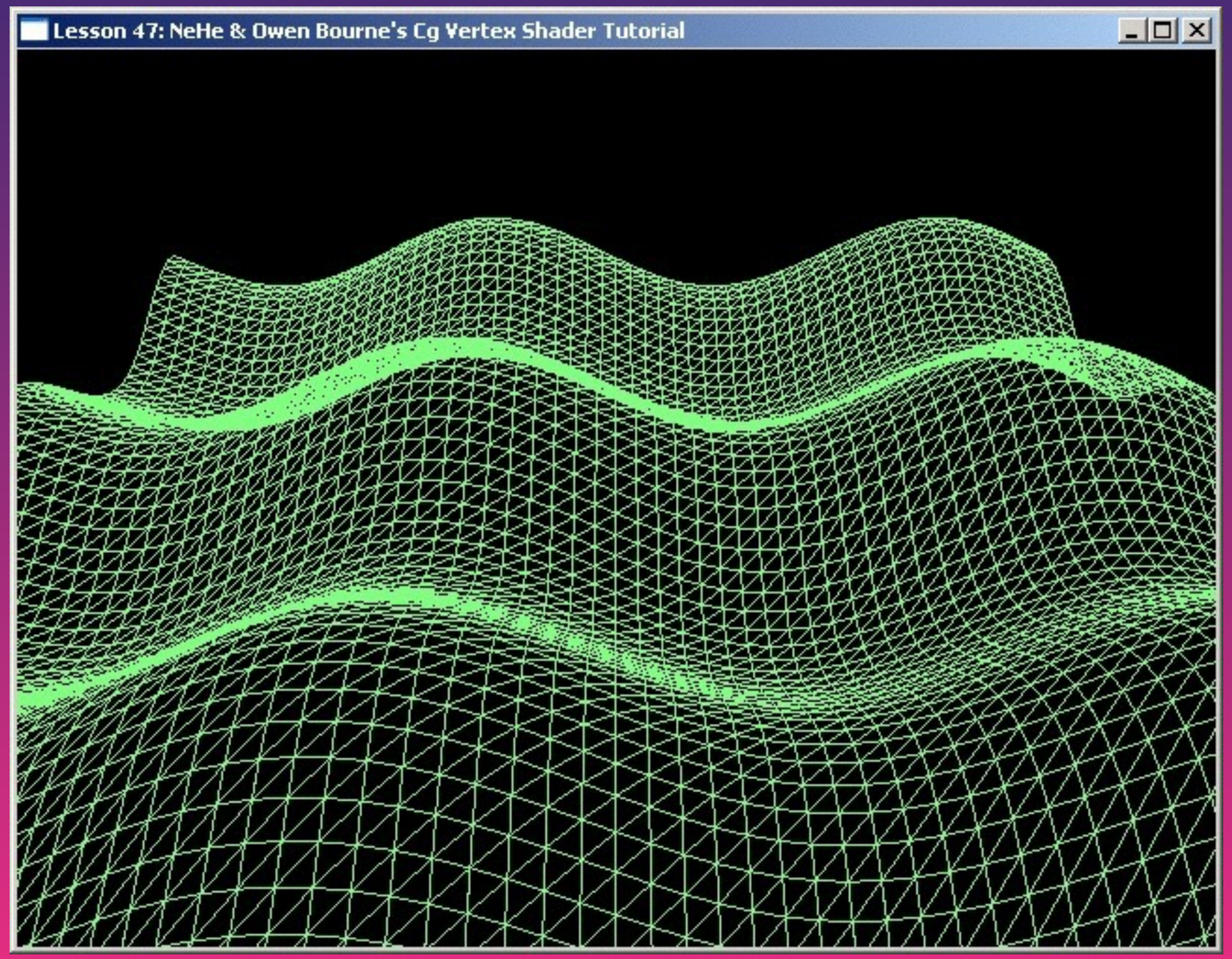
VIEWPORT MAPPING

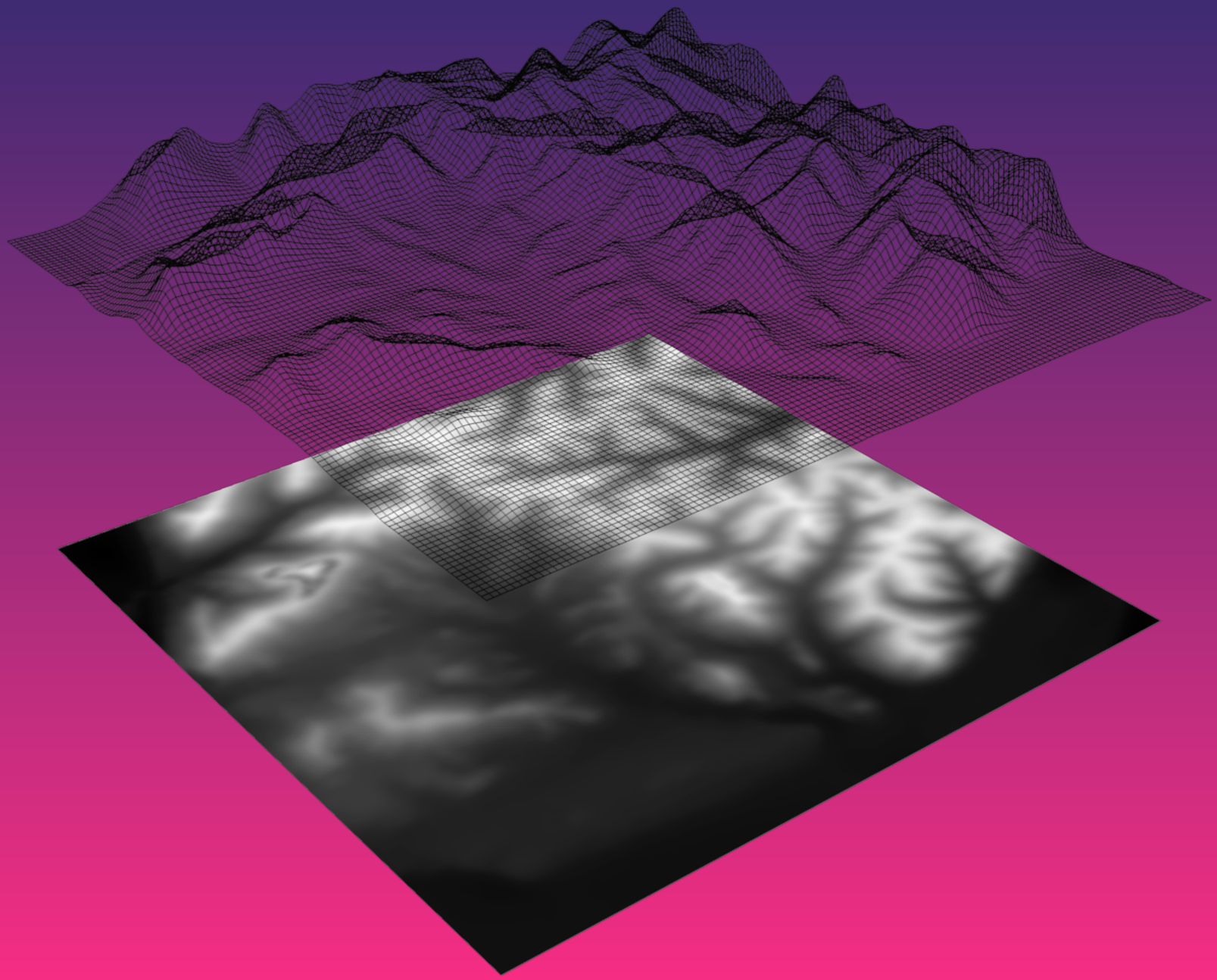
FRAGMENT SHADER

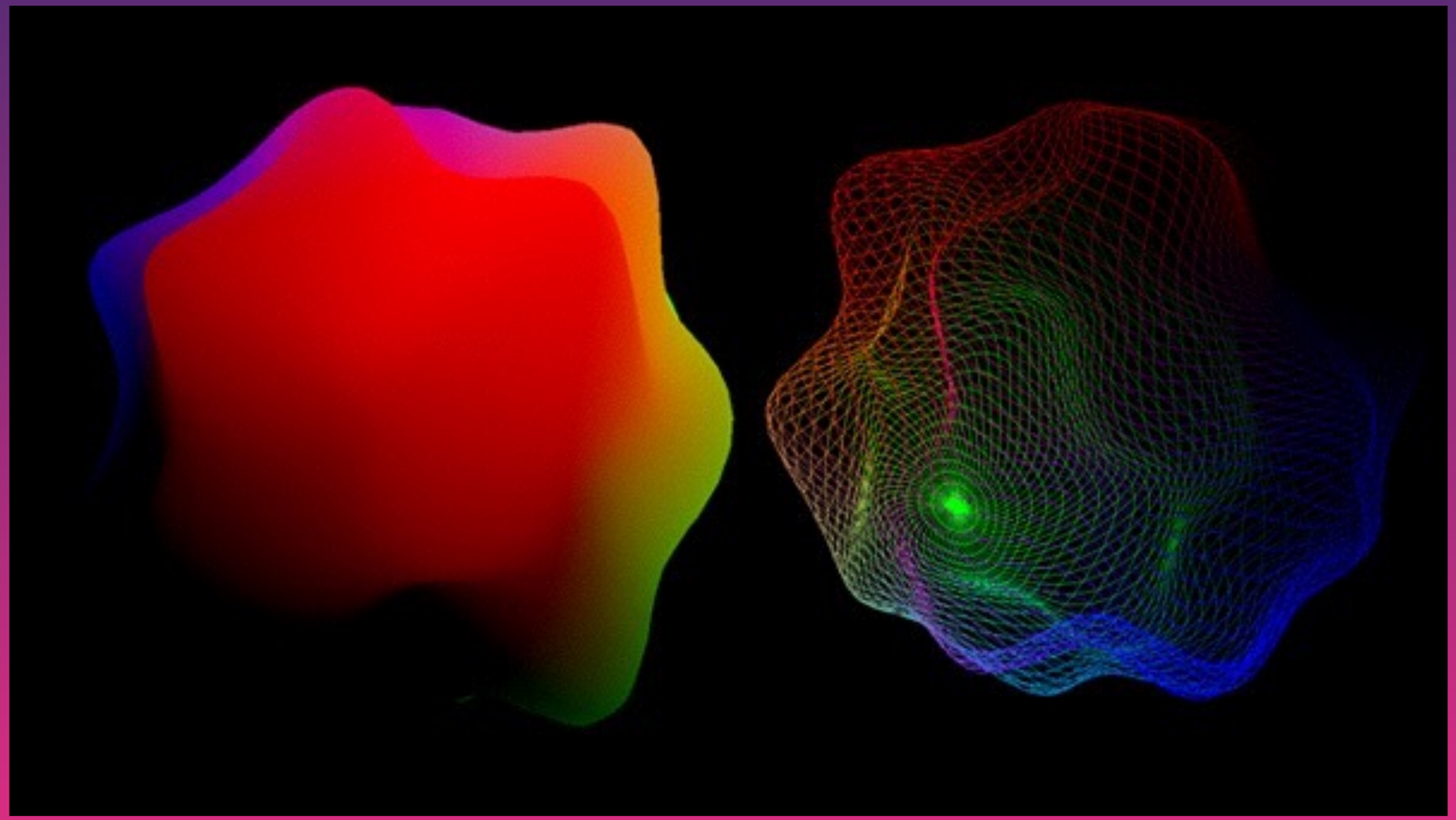


The Vertex Shader

A program that transforms the properties (such as position, color or others) of every vertex passed to the GPU.


Lesson 47: NeHe & Owen Bourne's Cg Vertex Shader Tutorial

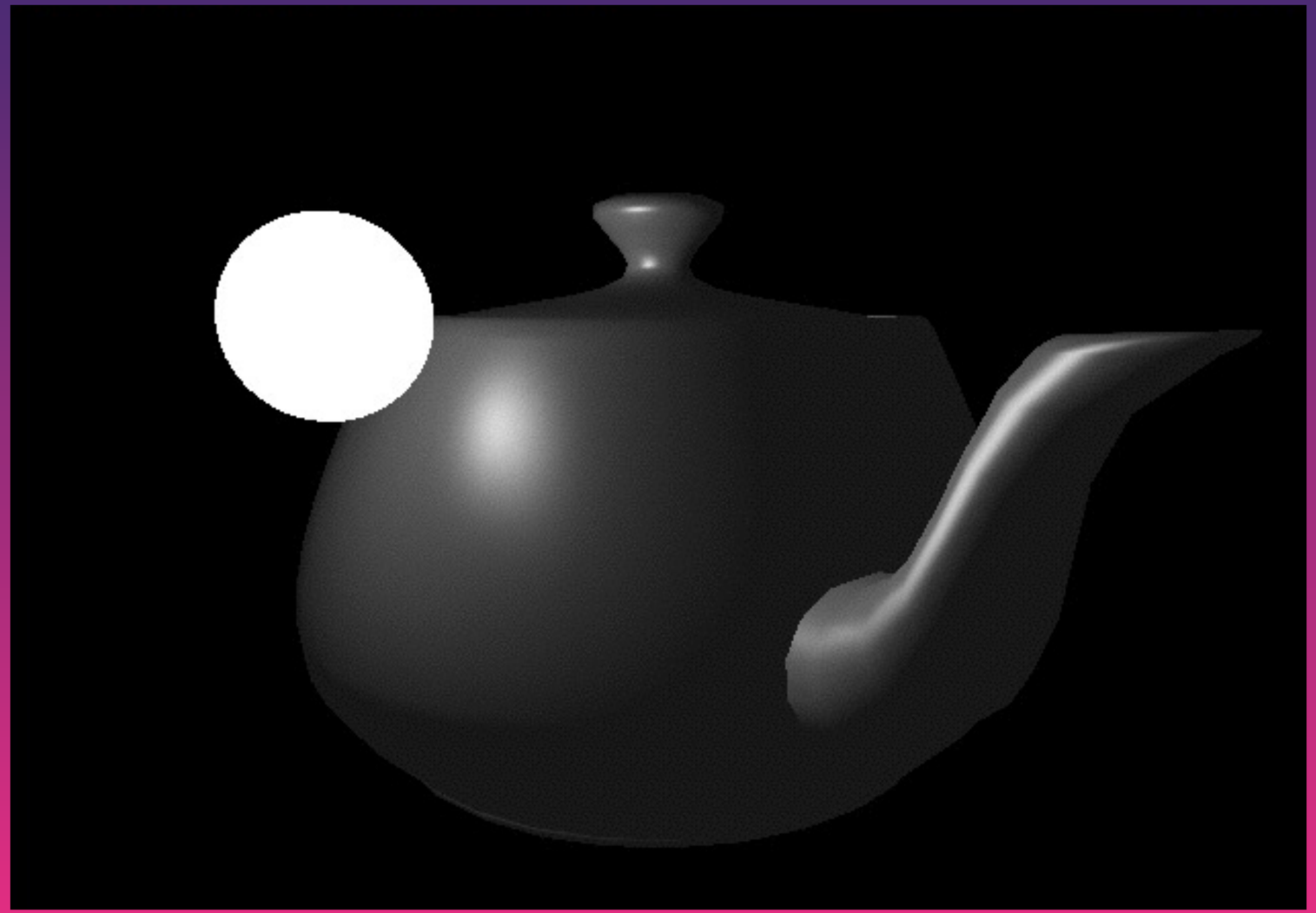


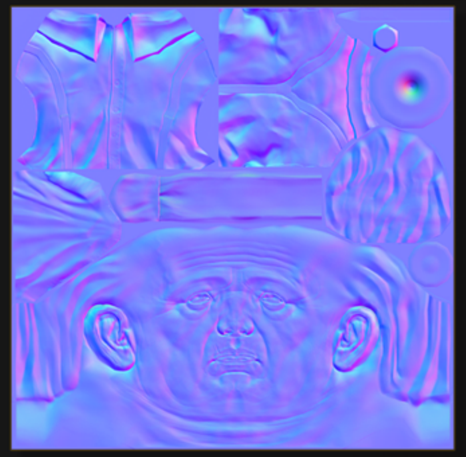


The vertex shader is where the modelview and projection matrices are applied to the vertex.

The Fragment Shader

A program that returns the color of each pixel when geometry is rasterized on the GPU.

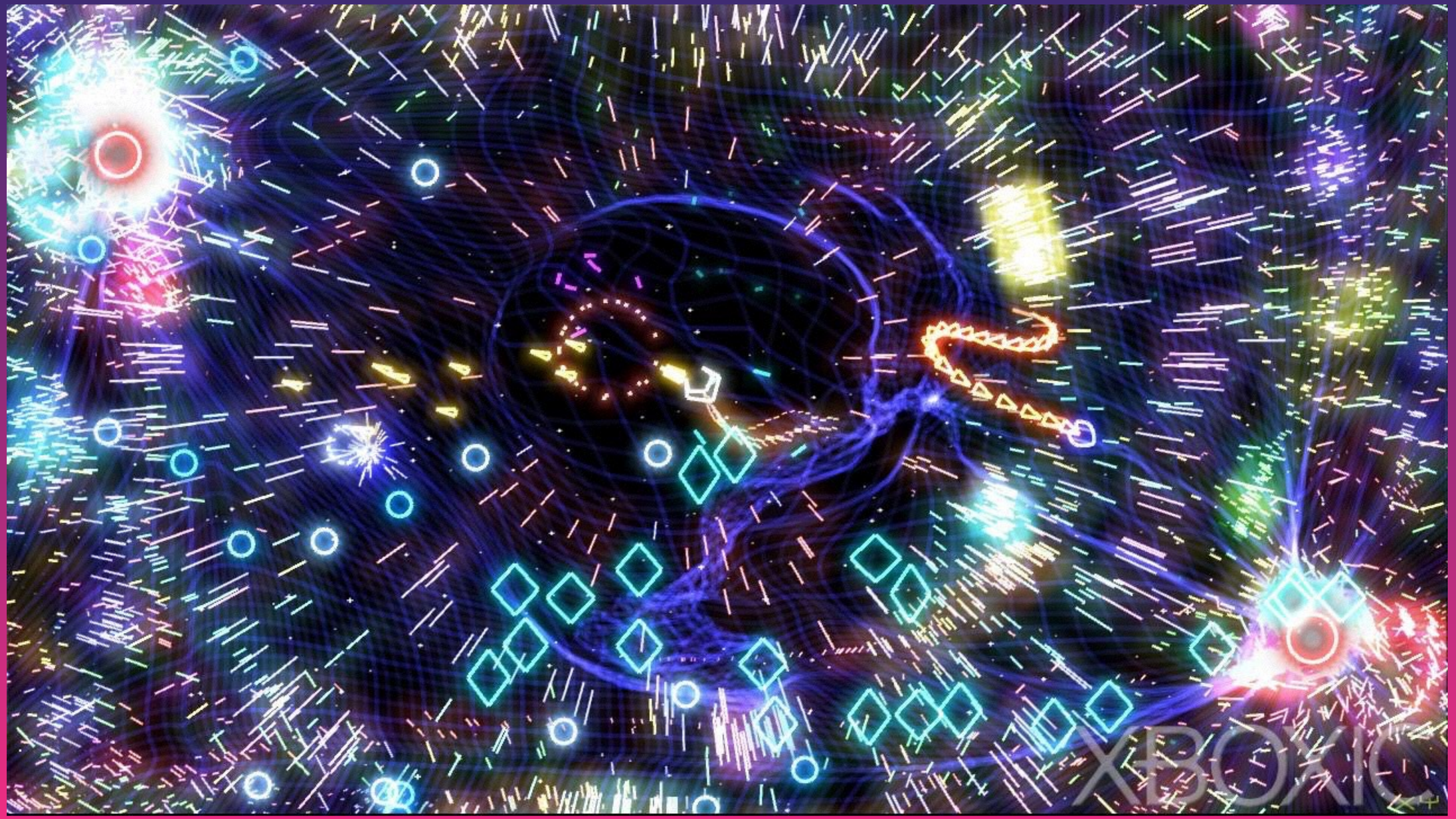




2048x2048 Textures
2754 tris

www.Alexander delagrange.com


4
4
4
$0


XBOXIC

The final shader program is a combination of a vertex and a fragment shader.

GLSL

(OpenGL Shading Language)

Anatomy of a simple GLSL vertex shader.

```
attribute vec4 position;
attribute vec2 texCoord;
```

ATTRIBUTES

```
uniform mat4 modelView;
uniform mat4 projection;
```

UNIFORMS

```
varying vec2 texCoordVar;
```

VARYING VARIABLES

```
void main()
{
    vec4 p = modelView * position;
    gl_Position = projection * p;
    texCoordVar = texCoord;
}
```

Anatomy of a simple GLSL fragment shader.

```
uniform sampler2D texture;
```

UNIFORMS

```
varying vec2 texCoordVar;
```

VARYING VARIABLES

```
void main()
{
    gl_FragColor = texture2D( texture, texCoordVar);
}
```

Setting up shaders in OpenGL.

Need to use GLEW library if you are on Windows.

Get it from:

http://glew.sourceforge.net/

The OpenGL Extension Wrangler Library

The OpenGL Extension Wrangler Library (GLEW) is a cross-platform open-source C/C++ extension loading library. GLEW provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform. OpenGL core and extension functionality is exposed in a single header file. GLEW has been tested on a variety of operating systems, including Windows, Linux, Mac OS X, FreeBSD, Irix, and Solaris.

Downloads

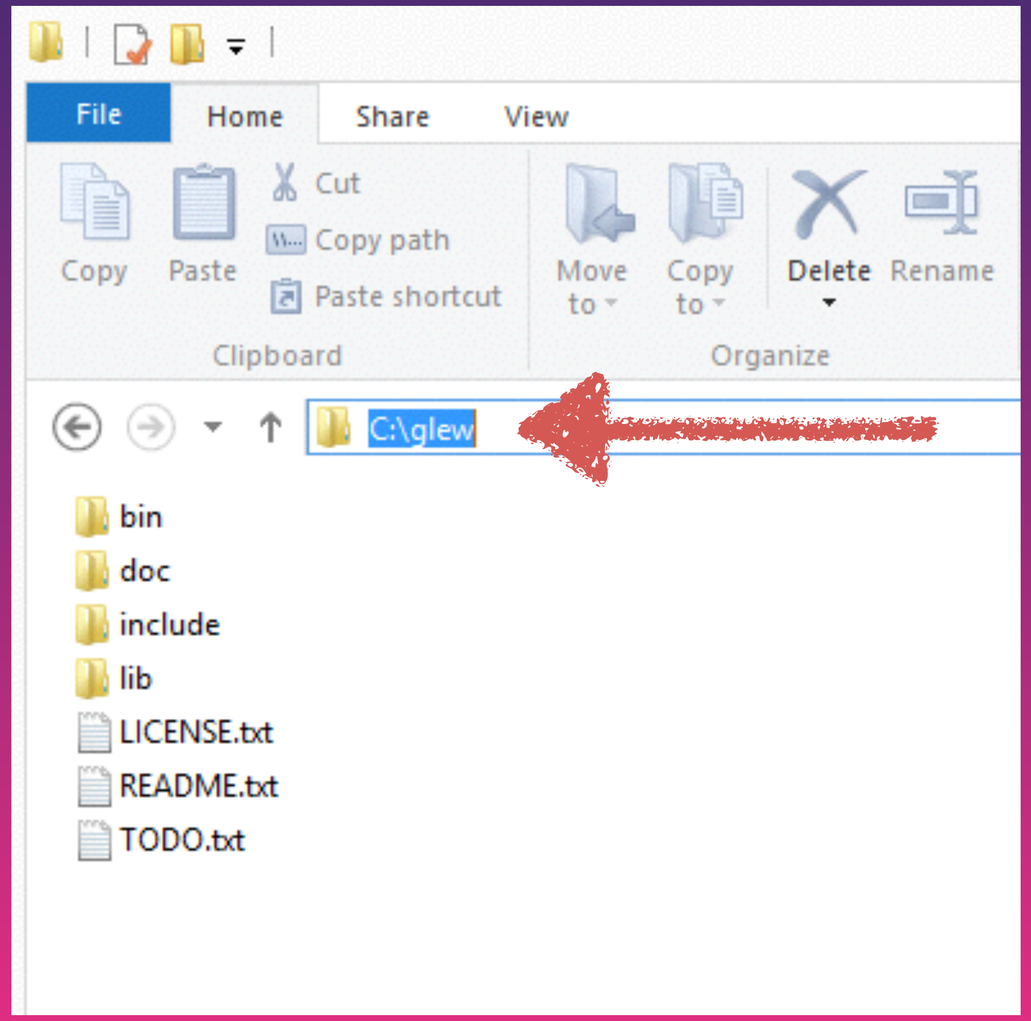
GLEW is distributed as source and precompiled binaries.
The latest release is 1.11.0[08-11-14]:

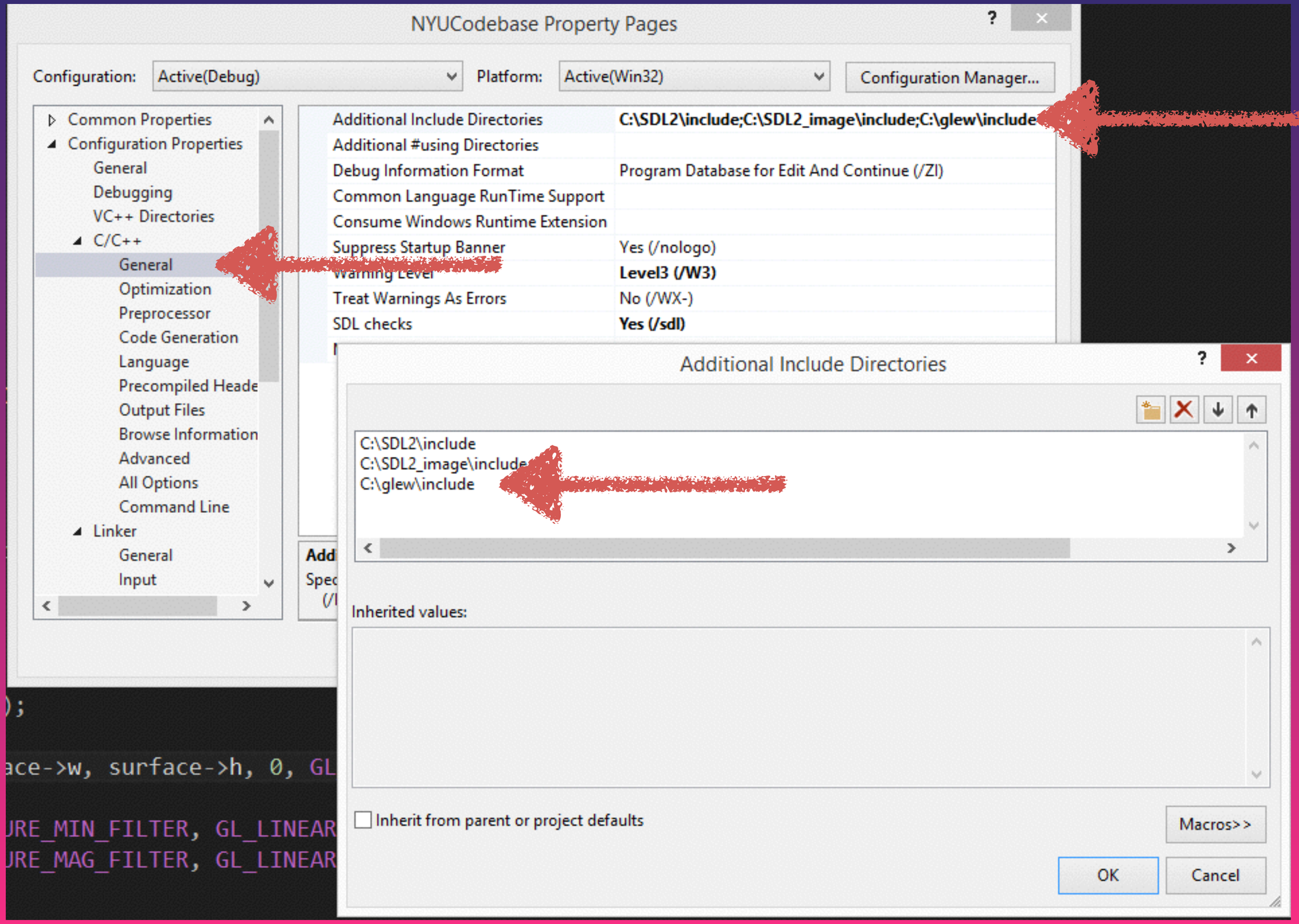
Source	ZIP | TGZ
Binaries	Windows 32-bit and 64-bit

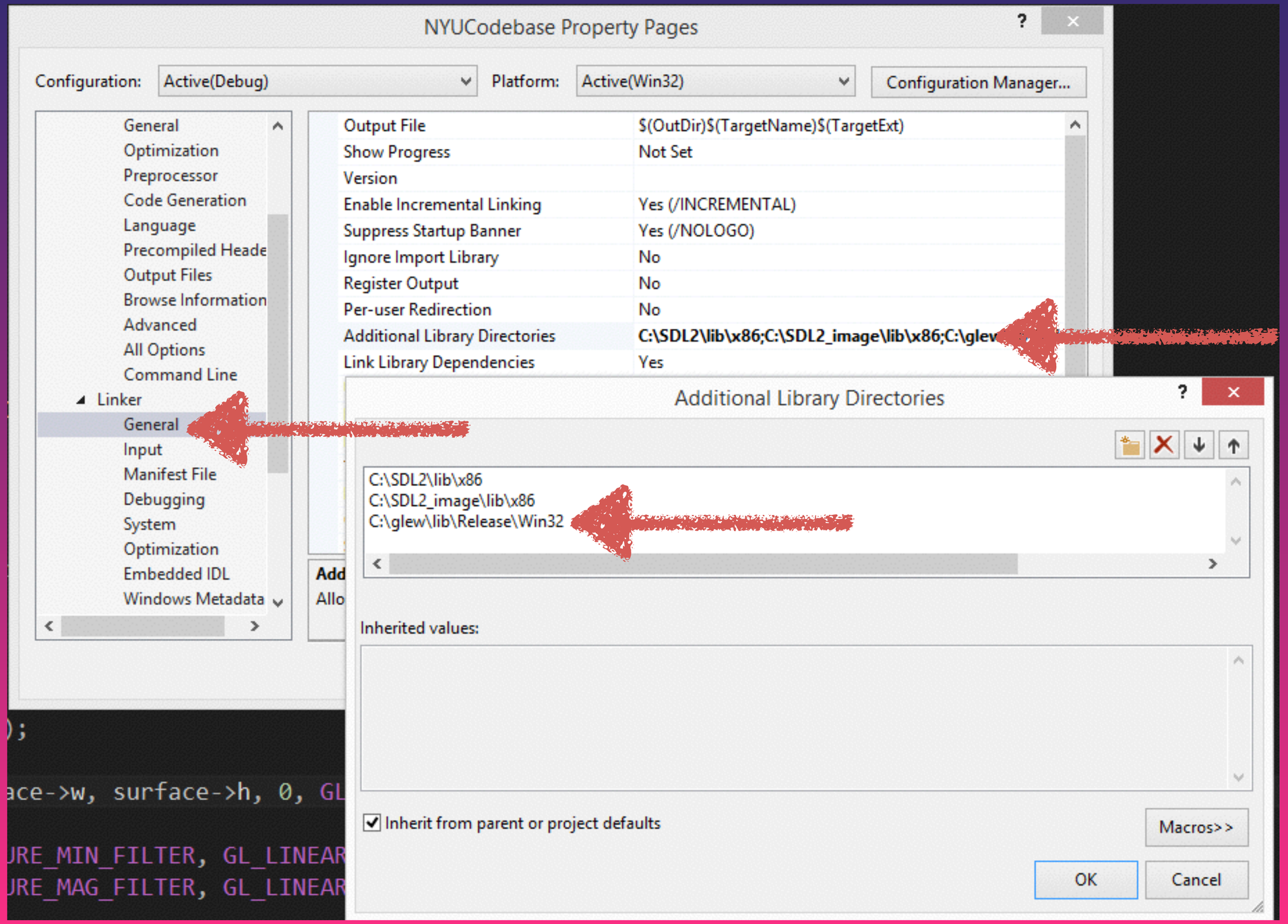
An up-to-date copy is also available using git:

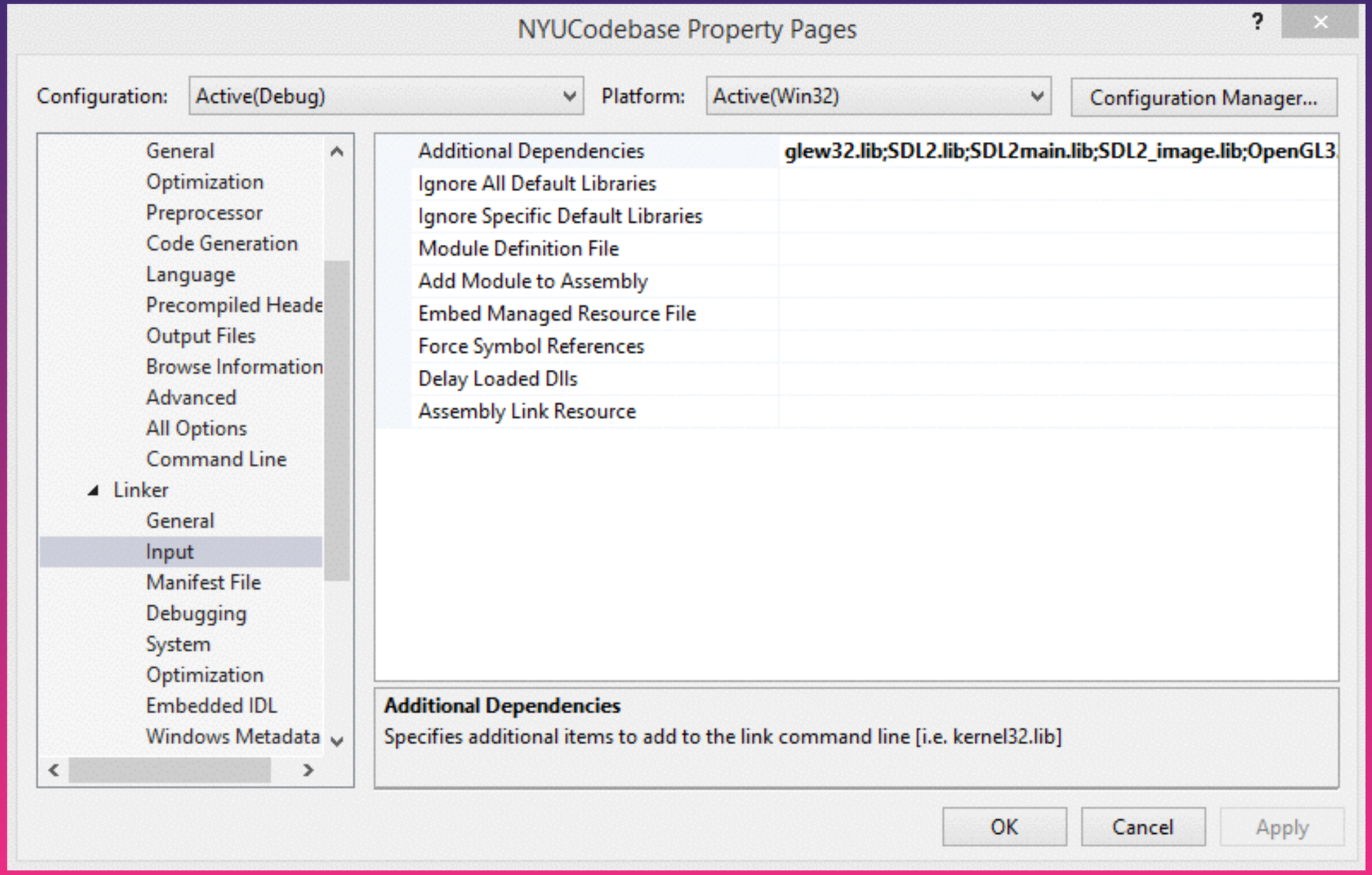
- github
  `git clone https://github.com/nigels-com/glew.git glew`
- Sourceforge
  `git clone git://git.code.sf.net/p/glew/code glew`

Unsupported snapshots are also available:


File
Home
Share
View
Cut
Copy path
Copy
Paste
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
C:\glew
bin
doc
include
lib
LICENSE.txt
README.txt
TODO.txt


NYUCodebase Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Common Properties
Configuration Properties
General
Debugging
VC++ Directories
C/C++
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Heade
Output Files
Browse Information
Advanced
All Options
Command Line
Linker
General
Input
Additional Include Directories
C:\SDL2\include;C:\SDL2_image\include;C:\glew\include
Additional #using Directories
Debug Information Format
Program Database for Edit And Continue (/ZI)
Common Language RunTime Support
Consume Windows Runtime Extension
Suppress Startup Banner
Yes (/nologo)
Level3 (/W3)
Treat Warnings As Errors
No (/WX-)
SDL checks
Yes (/sdl)
Additional Include Directories
C:\SDL2\include
C:\SDL2_image\include
C:\glew\include
Inherited values:
Inherit from parent or project defaults
Macros>>
OK
Cancel
ace->w, surface->h, 0, GL
URE_MIN_FILTER, GL_LINEAR
URE_MAG_FILTER, GL_LINEAR


NYUCodebase Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Heade
Output Files
Browse Information
Advanced
All Options
Command Line
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Windows Metadata
Output File $(OutDir)$(TargetName)$(TargetExt)
Show Progress Not Set
Version
Enable Incremental Linking Yes (/INCREMENTAL)
Suppress Startup Banner Yes (/NOLOGO)
Ignore Import Library No
Register Output No
Per-user Redirection No
Additional Library Directories C:\SDL2\lib\x86;C:\SDL2_image\lib\x86;C:\glev
Link Library Dependencies Yes
Additional Library Directories
C:\SDL2\lib\x86
C:\SDL2_image\lib\x86
C:\glew\lib\Release\Win32
Inherited values:
Inherit from parent or project defaults
Macros>>
OK
Cancel
);
ace->w, surface->h, 0, GL
URE_MIN_FILTER, GL_LINEAR
URE_MAG_FILTER, GL_LINEAR


NYUCodebase Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
General
Optimization
Preprocessor
Code Generation
Language
Precompiled Heade
Output Files
Browse Information
Advanced
All Options
Command Line
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Windows Metadata
Additional Dependencies
glew32.lib;SDL2.lib;SDL2main.lib;SDL2_image.lib;OpenGL3
Ignore All Default Libraries
Ignore Specific Default Libraries
Module Definition File
Add Module to Assembly
Embed Managed Resource File
Force Symbol References
Delay Loaded Dlls
Assembly Link Resource
Additional Dependencies
Specifies additional items to add to the link command line [i.e. kernel32.lib]
OK
Cancel
Apply

You'll need to copy glew32.dll from glew's bin/Release/Win32 folder to where your app DLLs are.

Include it.

```
#include <GL/glew.h>
```

Initialize it after creating the OpenGL context.

```
glewInit();
```

Rendering a textured sprite on the screen using shaders only.

Step One

Load and compile our vertex and fragment shaders.

```
attribute vec4 position;
attribute vec2 texCoord;

uniform mat4 modelView;
uniform mat4 projection;

varying vec2 texCoordVar;

void main()
{
    vec4 p = modelView * position;
    gl_Position = projection * p;
    texCoordVar = texCoord;
}
```

```
uniform sampler2D texture;

varying vec2 texCoordVar;

void main()
{
    gl_FragColor = texture2D( texture, texCoordVar);
}
```

```
std::string vertexShader =
"attribute vec4 position;\n\
attribute vec2 texCoord;\n\
uniform mat4 modelView;\n\
uniform mat4 projection;\n\
varying vec2 texCoordVar;\n\
void main()\n\
{\n\
    vec4 p = modelView * position;\n\
    gl_Position = projection * p;\n\
    texCoordVar = texCoord;\n\
}\n";
```

```
std::string fragmentShader =
"uniform sampler2D texture;\n\
varying vec2 texCoordVar;\n\
void main()\n\
{\n\
    gl_FragColor = texture2D( texture, texCoordVar);\n\
}\n";
```

Creating and compiling a shader.

Use GL_VERTEX_SHADER for vertex shader.

```
GLuint vertexShaderID = glCreateShader(GL_VERTEX_SHADER);

const char *vertexShaderString = vertexShader.c_str();
int vertexShaderStringLength = vertexShader.size();

glShaderSource(vertexShaderID, 1, &vertexShaderString, & vertexShaderStringLength);
glCompileShader(vertexShaderID);
```

Use GL_FRAGMENT_SHADER for fragment shader.

```
GLuint fragmentShaderID = glCreateShader(GL_FRAGMENT_SHADER);

const char *fragmentShaderString = fragmentShader.c_str();
int fragmentShaderStringLength = fragmentShader.size();

glShaderSource(fragmentShaderID, 1, &fragmentShaderString, &fragmentShaderStringLength);
glCompileShader(fragmentShaderID);
```

Step Two

Create and link the shader program from the fragment and vertex shaders.

```
GLuint exampleProgram = glCreateProgram();
glAttachShader(exampleProgram, vertexShaderID);
glAttachShader(exampleProgram, fragmentShaderID);
glLinkProgram(exampleProgram);
```

Step Three

Get references to the attribute and uniform locations of the shader program so we can pass data to it.

```
attribute vec4 position;
attribute vec2 texCoord;
```

ATTRIBUTES

```
uniform mat4 modelView;
uniform mat4 projection;
```

UNIFORMS

```
uniform sampler2D texture;
```

UNIFORMS

Uniforms

```
GLint projectionMatrixUniform = glGetUniformLocation(exampleProgram, "projection");
GLint modelviewMatrixUniform = glGetUniformLocation(exampleProgram, "modelView");
```

Attributes

```
GLuint positionAttribute = glGetAttribLocation(exampleProgram, "position");
GLuint texCoordAttribute = glGetAttribLocation(exampleProgram, "texCoord");
```

Drawing using our shader program.

Manual matrices for projection and modelview.

Manual ortho projection matrix.

```
Matrix projectionMatrix;

float l = -1.331f;
float r = 1.33f;
float t = 1.0f;
float b = -1.0f;
float f = 1.0f;
float n = -1.0f;

projectionMatrix.identity();
projectionMatrix.m[0][0] = 2.0/(r-l);
projectionMatrix.m[1][1] = 2.0/(t-b);
projectionMatrix.m[2][2] = -2.0/(f-n);
```

Tell OpenGL to use our program.

```
glUseProgram(exampleProgram);
```

Bind our modelview and projection matrices.

```
glUniformMatrix4fv(projectionMatrixUniform, 1, GL_FALSE, projectionMatrix.ml);
glUniformMatrix4fv(modelviewMatrixUniform, 1, GL_FALSE, modelviewMatrix.ml);
```

When drawing, we now pass our vertex data as attributes.

Old way:

```
float vertices[] = {-w/2.0f, h/2.0f, -w/2.0f, -h/2.0f, w/2.0f, -h/2.0f, w/2.0f, h/2.0f};
glVertexPointer(2, GL_FLOAT, 0, vertices);
glEnableClientState(GL_VERTEX_ARRAY);
```

New way using the attribute location we got for vertex position.

```
float vertices[] = {-w/2.0f, h/2.0f, -w/2.0f, -h/2.0f, w/2.0f, -h/2.0f, w/2.0f, h/2.0f};
glVertexAttribPointer(positionAttribute, 2, GL_FLOAT, false, 0, vertices);
glEnableVertexAttribArray(positionAttribute);
```

Same for UV coordinates.

Old way:

```
float uvs[] = {0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0};
glTexCoordPointer(2, GL_FLOAT, 0, uvs);
glEnableClientState(GL_TEXTURE_COORD_ARRAY);
```

New way using the attribute location we got for texture coordinate

```
float uvs[] = {0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 0.0};
glVertexAttribPointer(texCoordAttribute, 2, GL_FLOAT, false, 0, uvs);
glEnableVertexAttribArray(texCoordAttribute);
```

Draw as usual.

```
unsigned int indices[] = {0, 1, 2, 0, 2, 3};
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, indices);
```

And disable attribute arrays just like we disabled client states before.

```
glDisableVertexAttribArray(positionAttribute);
glDisableVertexAttribArray(texCoordAttribute);
```

To recap.

Creating a shader program.

- Load and compile fragment and vertex shaders.
- Create and link shader program from the fragment and vertex shaders.
- Get references to the uniform and attribute locations in the shader program.

Drawing using a shader program.

- Setup our projection and modelview matrices.
- Tell OpenGL to use our program.
- Bind our modelview and projection matrices to the appropriate uniform locations.
- Pass our vertex data using the appropriate attribute locations.
- Draw as usual.

Catching GLSL errors when compiling shaders.

After calling glCompileShader you can check if it failed to compile and get a readable error message.

```
GLint compileSuccess;
glGetShaderiv(vertexShaderID, GL_COMPILE_STATUS, &compileSuccess);
if (compileSuccess == GL_FALSE) {

        GLchar messages[256];
        glGetShaderInfoLog(shaderID, sizeof(messages), 0, &messages[0]);
        std::cout << messages << std::endl;

}
```

Intro to basic GLSL

```
attribute vec4 position;
attribute vec2 texCoord;

uniform mat4 modelView;
uniform mat4 projection;

varying vec2 texCoordVar;

void main()
{
    vec4 p = modelView * position;
    gl_Position = projection * p;
    texCoordVar = texCoord;
}
```

```
uniform sampler2D texture;

varying vec2 texCoordVar;

void main()
{
    gl_FragColor = texture2D( texture, texCoordVar);
}
```

Basic Types

void	no function return value or empty parameter list
bool	Boolean
int	signed integer
float	floating scalar
vec2, vec3, vec4	n-component floating point vector
bvec2, bvec3, bvec4	Boolean vector
ivec2, ivec3, ivec4	signed integer vector
mat2, mat3, mat4	2x2, 3x3, 4x4 float matrix
sampler2D	access a 2D texture
samplerCube	access cube mapped texture

Vertex shader must set **gl_Position** (which is a vec4) to set the final position of the vertex being drawn.

If it is passing any “varying” data to the pixel shader, it must set those as well.

```
attribute vec4 position;
attribute vec2 texCoord;

uniform mat4 modelView;
uniform mat4 projection;

varying vec2 texCoordVar;

void main()
{
    vec4 p = modelView * position;
    gl_Position = projection * p;
    texCoordVar = texCoord;
}
```

Fragment shader must set **gl_FragColor** (which is a vec4) to set the final color of the pixel being rendered (in RGBA format).

The texture2D function is built in and takes a sampler2D texture and a vec2 UV coordinate and returns a vec4 RGBA color from that texture at that coordinate.

```
uniform sampler2D texture;

varying vec2 texCoordVar;

void main()
{
    gl_FragColor = texture2D( texture, texCoordVar);
}
```

Simple shader examples.

Inverting texture color.

```
uniform sampler2D texture;

varying vec2 texCoordVar;

void main()
{
    vec4 finalColor = 1.0 - texture2D( texture, texCoordVar);
    finalColor.a = texture2D( texture, texCoordVar).a;
    gl_FragColor = finalColor;
}
```

Making a texture black and white.

```
uniform sampler2D texture;

varying vec2 texCoordVar;

void main()
{
    vec4 finalColor = vec4((texture2D( texture, texCoordVar).x +
texture2D( texture, texCoordVar).y + texture2D( texture, texCoordVar).z)/3.0);

    finalColor.a = texture2D( texture, texCoordVar).a;
    gl_FragColor = finalColor;
}
```

Functions in GLSL

Built-In Functions

Angle & Trigonometry Functions [8.1]

Component-wise operation. Parameters specified as *angle* are assumed to be in units of radians. T is float, vec2, vec3, vec4.

T **radians**(T *degrees*)	degrees to radians
T **degrees**(T *radians*)	radians to degrees
T **sin**(T *angle*)	sine
T **cos**(T *angle*)	cosine
T **tan**(T *angle*)	tangent
T **asin**(T *x*)	arc sine
T **acos**(T *x*)	arc cosine
T **atan**(T *y*, T *x*) T **atan**(T *y_over_x*)	arc tangent

Exponential Functions [8.2]

Component-wise operation. T is float, vec2, vec3, vec4.

T **pow**(T *x*, T *y*)	x^y
T **exp**(T *x*)	e^x
T **log**(T *x*)	ln
T **exp2**(T *x*)	2^x
T **log2**(T *x*)	$\log_2$
T **sqrt**(T *x*)	square root
T **inversesqrt**(T *x*)	inverse square root

Common Functions [8.3]

Component-wise operation. T is float, vec2, vec3, vec4.

T **abs**(T *x*)	absolute value
T **sign**(T *x*)	returns -1.0, 0.0, or 1.0
T **floor**(T *x*)	nearest integer <= *x*
T **ceil**(T *x*)	nearest integer >= *x*
T **fract**(T *x*)	*x* - **floor**(*x*)
T **mod**(T *x*, T *y*) T **mod**(T *x*, float *y*)	modulus
T **min**(T *x*, T *y*) T **min**(T *x*, float *y*)	minimum value
T **max**(T *x*, T *y*) T **max**(T *x*, float *y*)	maximum value
T **clamp**(T *x*, T *minVal*, T *maxVal*) T **clamp**(T *x*, float *minVal*, float *maxVal*)	**min**(**max**(*x*, **minVal**), **maxVal**)
T **mix**(T *x*, T *y*, T *a*) T **mix**(T *x*, T *y*, float *a*)	linear blend of *x* and *y*
T **step**(T *edge*, T *x*) T **step**(float *edge*, T *x*)	0.0 if *x* < *edge*, else 1.0
T **smoothstep**(T *edge0*, T *edge1*, T *x*) T **smoothstep**(float *edge0*, float *edge1*, T *x*)	clip and smooth

Geometric Functions [8.4]

These functions operate on vectors as vectors, not component-wise. T is float, vec2, vec3, vec4.

float **length**(T *x*)	length of vector
float **distance**(T *p0*, T *p1*)	distance between points
float **dot**(T *x*, T *y*)	dot product
vec3 **cross**(vec3 *x*, vec3 *y*)	cross product
T **normalize**(T *x*)	normalize vector to length 1
T **faceforward**(T *N*, T *I*, T *Nref*)	returns *N* if **dot**(*Nref*, *I*) < 0, else -*N*
T **reflect**(T *I*, T *N*)	reflection direction *I* - 2 * **dot**(*N*,*I*) * *N*
T **refract**(T *I*, T *N*, float *eta*)	refraction vector

Matrix Functions [8.5]

Type mat is any matrix type.

mat **matrixCompMult**(mat *x*, mat *y*)	multiply *x* by *y* component-wise

Vector Relational Functions [8.6]

Compare *x* and *y* component-wise. Sizes of input and return vectors for a particular call must match. Type bvec is bvec*n*; vec is vec*n*; ivec is ivec*n* (where *n* is 2, 3, or 4). T is the union of vec and ivec.

bvec **lessThan**(T *x*, T *y*)	x < y
bvec **lessThanEqual**(T *x*, T *y*)	x <= y
bvec **greaterThan**(T *x*, T *y*)	x > y
bvec **greaterThanEqual**(T *x*, T *y*)	x >= y
bvec **equal**(T *x*, T *y*) bvec **equal**(bvec *x*, bvec *y*)	x == y
bvec **notEqual**(T *x*, T *y*) bvec **notEqual**(bvec *x*, bvec *y*)	x!= y
bool **any**(bvec *x*)	true if any component of *x* is true
bool **all**(bvec *x*)	true if all components of *x* are true
bvec **not**(bvec *x*)	logical complement of *x*

Texture Lookup Functions [8.7]

Available only in vertex shaders.

vec4 **texture2DLod**(sampler2D *sampler*, vec2 *coord*, float *lod*)
vec4 **texture2DProjLod**(sampler2D *sampler*, vec3 *coord*, float *lod*)
vec4 **texture2DProjLod**(sampler2D *sampler*, vec4 *coord*, float *lod*)
vec4 **textureCubeLod**(samplerCube *sampler*, vec3 *coord*, float *lod*)

Available only in fragment shaders.

vec4 **texture2D**(sampler2D *sampler*, vec2 *coord*, float *bias*)
vec4 **texture2DProj**(sampler2D *sampler*, vec3 *coord*, float *bias*)
vec4 **texture2DProj**(sampler2D *sampler*, vec4 *coord*, float *bias*)
vec4 **textureCube**(samplerCube *sampler*, vec3 *coord*, float *bias*)

Available in vertex and fragment shaders.

vec4 **texture2D**(sampler2D *sampler*, vec2 *coord*)
vec4 **texture2DProj**(sampler2D *sampler*, vec3 *coord*)
vec4 **texture2DProj**(sampler2D *sampler*, vec4 *coord*)
vec4 **textureCube**(samplerCube *sampler*, vec3 *coord*)

Saturation function example.

```
uniform sampler2D texture;
varying vec2 texCoordVar;

vec3 czm_saturation(vec3 rgb, float adjustment)
{
    const vec3 W = vec3(0.2125, 0.7154, 0.0721);
    vec3 intensity = vec3(dot(rgb, W));
    return mix(intensity, rgb, adjustment);
}

void main()
{
    vec4 finalColor;
    finalColor.xyz = czm_saturation(texture2D( texture, texCoordVar).xyz, 2.0);
    finalColor.a = texture2D( texture, texCoordVar).a;
    gl_FragColor = finalColor;
}
```

Passing variables as uniforms.

Example: passing the saturation value from our C++ code.

```
uniform sampler2D texture;
uniform float saturationAmount;
varying vec2 texCoordVar;

vec3 czm_saturation(vec3 rgb, float adjustment)
{
    const vec3 W = vec3(0.2125, 0.7154, 0.0721);
    vec3 intensity = vec3(dot(rgb, W));
    return mix(intensity, rgb, adjustment);
}

void main()
{
    vec4 finalColor;
    finalColor.xyz = czm_saturation(texture2D( texture, texCoordVar).xyz, saturationAmount);
    finalColor.a = texture2D( texture, texCoordVar).a;
    gl_FragColor = finalColor;
}
```

Get the saturation amount uniform location.

```
GLint saturationAmountUniform = glGetUniformLocation(exampleProgram, "saturationAmount");
```

Before rendering, bind a value to it.

```
glUniform1f(saturationAmountUniform, fabs(sin(ticks * 2.0f)));
```

Some GLSL types and their corresponding C++ uniform binding functions.

float - glUniform1f(location, value);
vec2 - glUniform1f(location, value1, value2);
vec3 - glUniform3f(location, value1, value2, value3);
vec4 - glUniform4f(location, value1, value2, value3, value4);

Example: scrolling a texture.

```
uniform sampler2D texture;
uniform vec2 scroll;
varying vec2 texCoordVar;

void main()
{
    gl_FragColor = texture2D( texture, texCoordVar + scroll);
}
```

Get the scroll amount uniform location.

```
GLint scrollUniform = glGetUniformLocation(exampleProgram, "scroll");
```

Before rendering, bind a value to it.

```
glUniform2f(scrollUniform, ticks, 0.0f);
```

OpenGL ES2 and OpenGL ES2 shading language reference card:

https://www.khronos.org/opengles/sdk/docs/reference_cards/OpenGL-ES-2_0-Reference-card.pdf