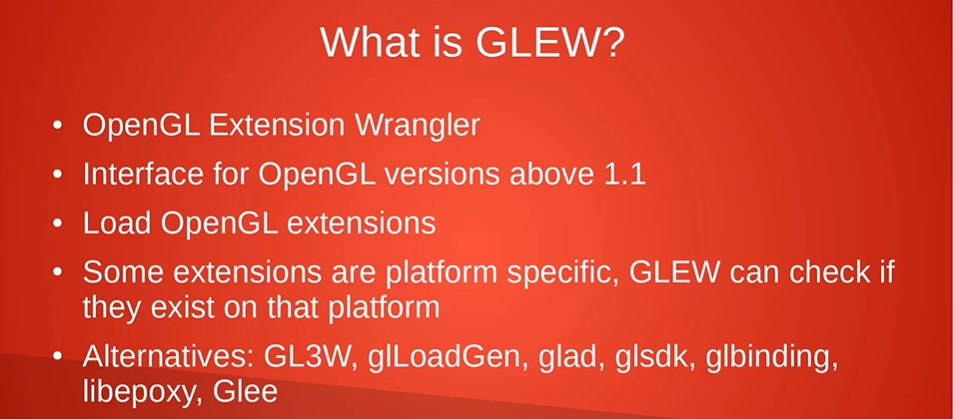
**GLEW : OpenGL Extension Wrangler**



Most compilers give you an OpenGL header that provides support for OpenGL 1.1, but most of the functions provided in that version of OpenGL have been long since deprecated, and lack important features such as shaders and vertex buffer objects. GLEW is basically just a header (there is a source file that must also be compiled, but you can just include it in your project) that provides newer OpenGL functionality.

1. What does it mean by extension?

[Here's a link from OpenGL.org about them.](https://www.opengl.org/wiki/OpenGL_Extension) They're just additional functions for OpenGL that aren't part of OpenGL. They provide functionality which may be useful, but you cannot guarantee that they're going to work on every computer since they are not part of core OpenGL.

1. What does it mean to load pointers to OpenGL functions?

OpenGL is implemented by the graphics driver as a bunch of functions implemented according to the spec, not as a standalone library, therefore cannot be simply linked to your project.

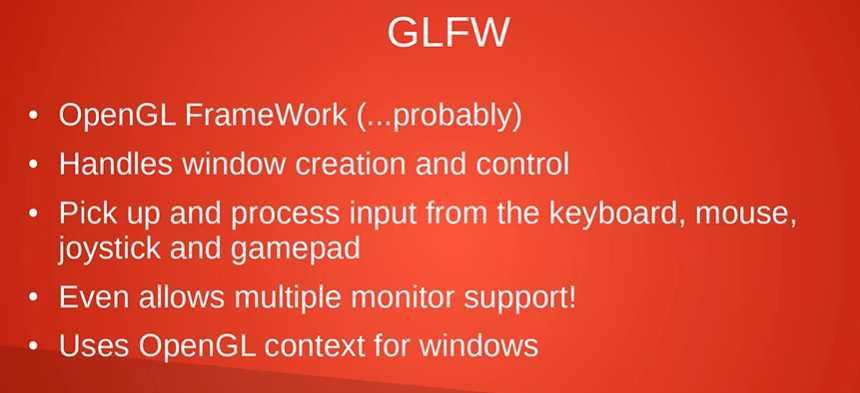
You have to call a function (such as GetProcAddress, but it depends on the platform you're developing for) to get the function pointer at runtime from the driver for each OpenGL function.

GLEW exists because doing just that can be very difficult. It's much easier when you can just include a header and call one function during initialization, which is exactly what GLEW does. Plus, GLEW is cross-platform.

1. What does it mean by "core as well as extensions"?

It means it supplies OpenGL functions as well as OpenGL extensions.

**GLFW**

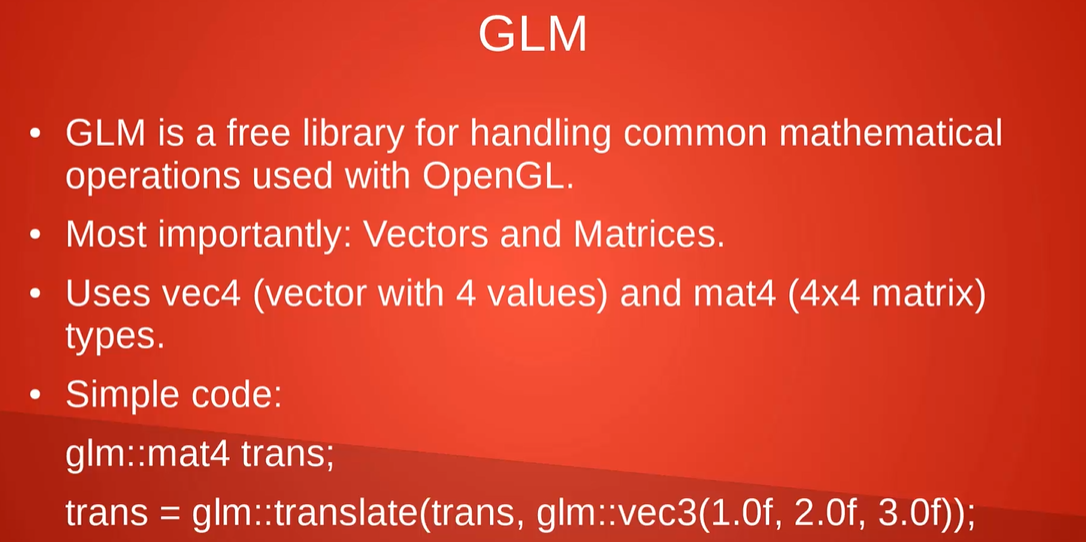


A possible reason that libraries like GLFW are needed is that OpenGL by itself does not provide any mechanisms for creating the necessary context, managing windows, user input, timing etc. There are several other libraries available for aiding OpenGL development.

**SDL**



**GLM**

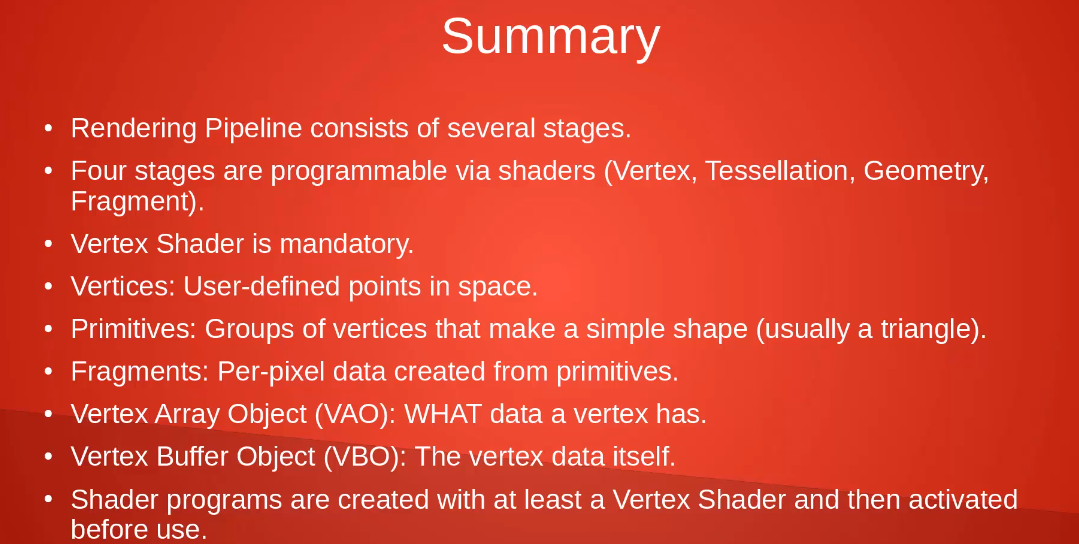


Culling -> 剔除

Rasterize -> 栅格化

Interpolate -> 插入

Stencil -> 模板

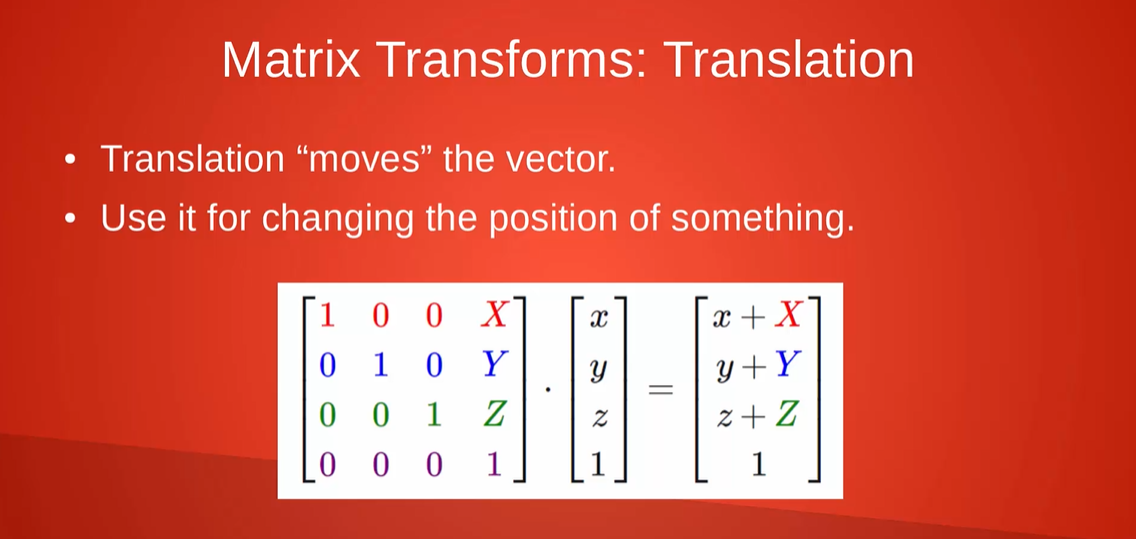
Vertex -> Vertex shader

Primitives -> Geometry shader

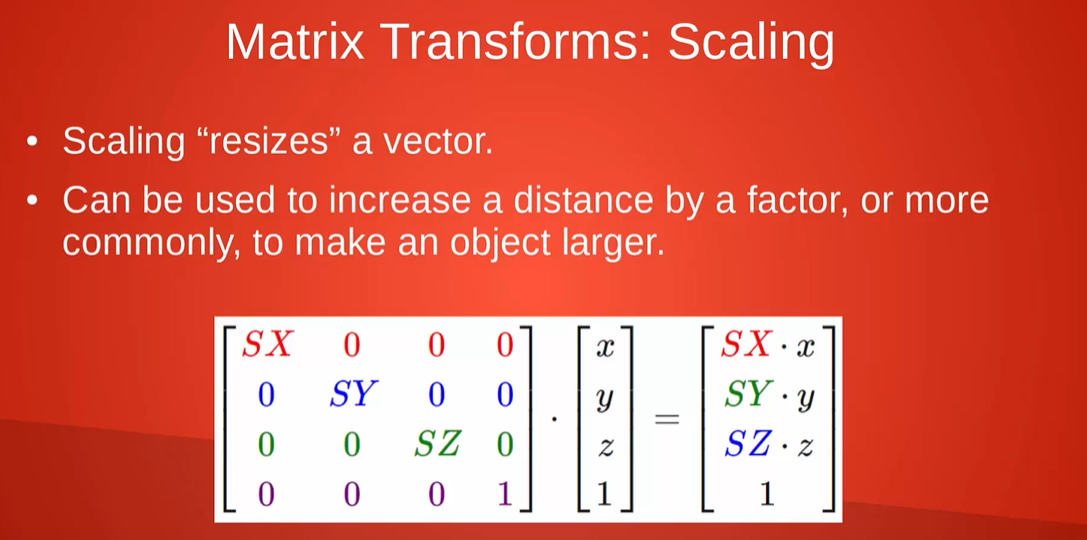
Fragments -> Fragment shader

Attribute Pointers define where and how shaders can access vertex data.

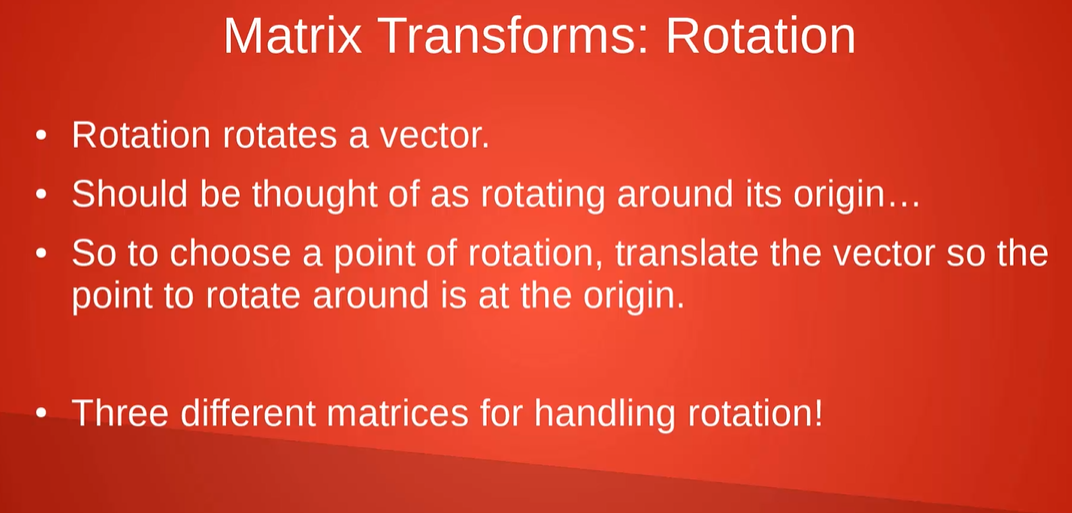
**Transforming: Translations**

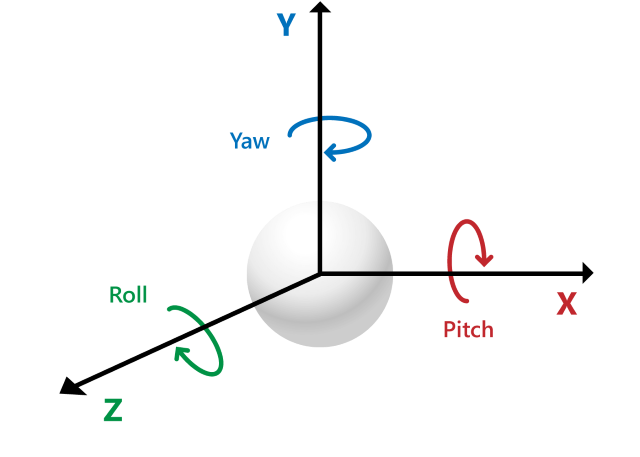
****

**Transforming: Scaling**

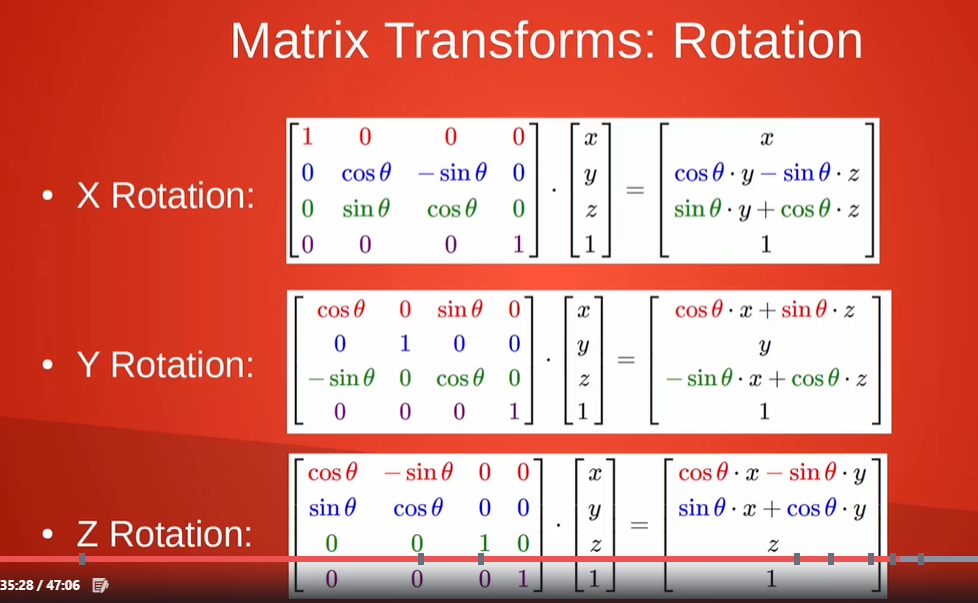
****

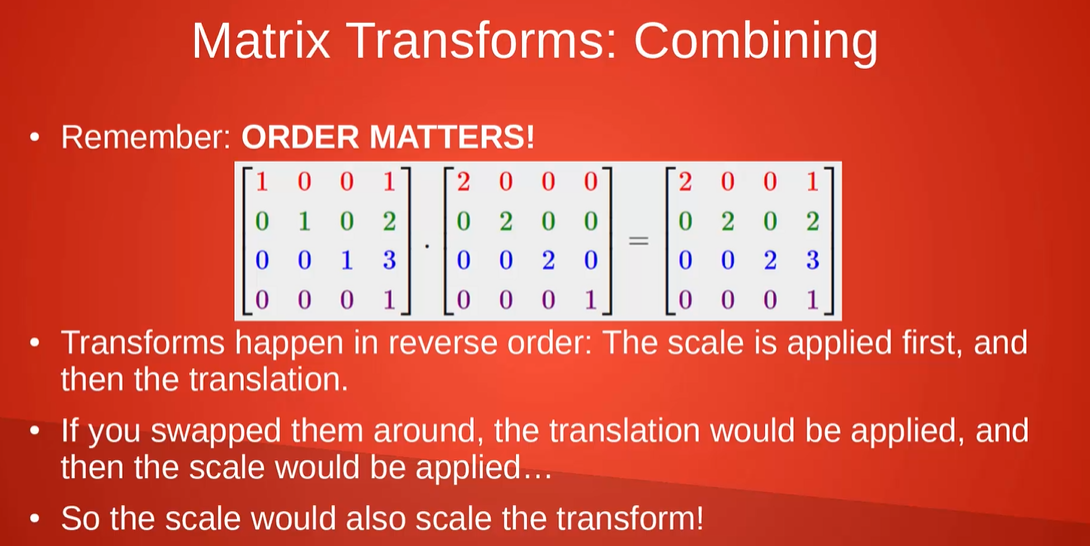
**Transforming: Rotation**

****

****

**!!!!!!! Z axis here is toward screen in computer graphic**

****

****

# The Model, View and Projection matrices

**http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/**

**The radian**

**one radian is**

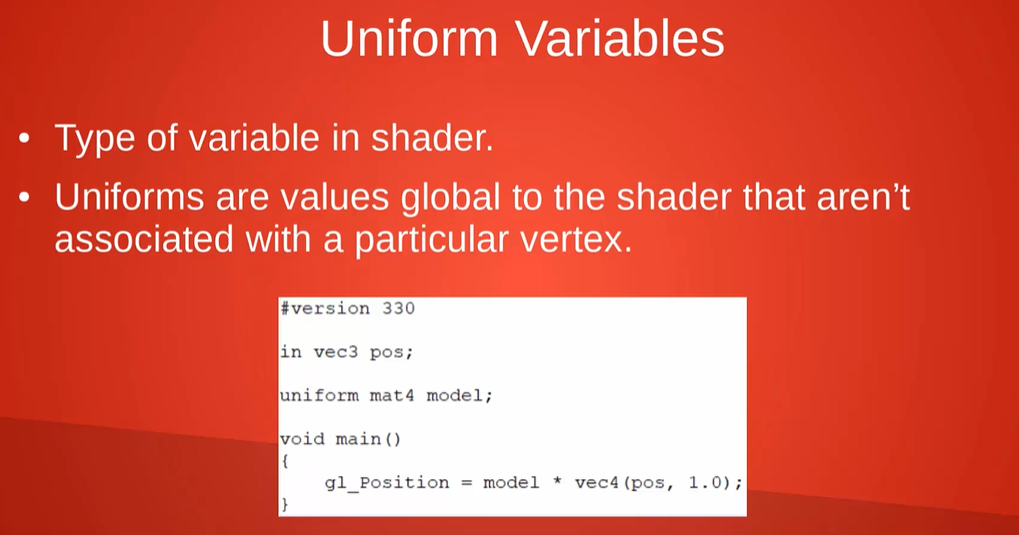
**180**

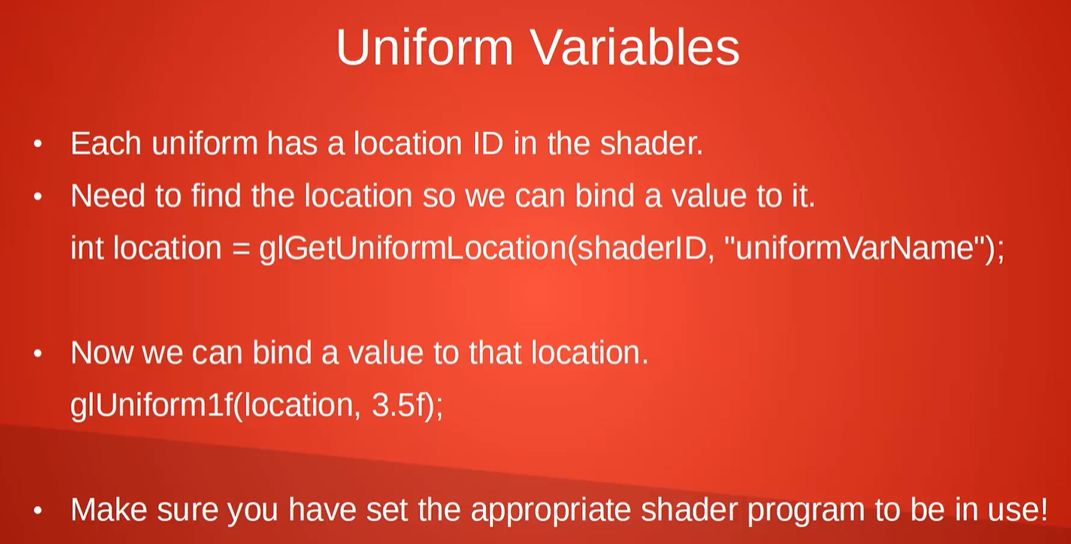
**/**

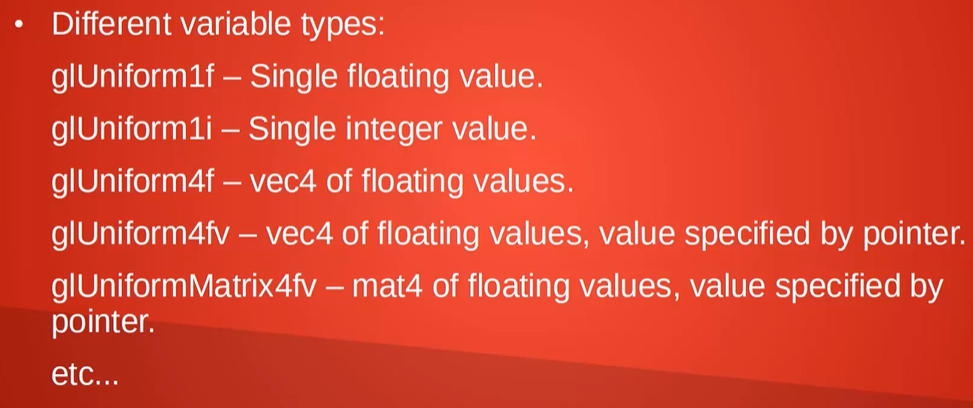
***π***

[**degrees**](https://en.wikipedia.org/wiki/Degree_(angle)) **or just under 57.3°**

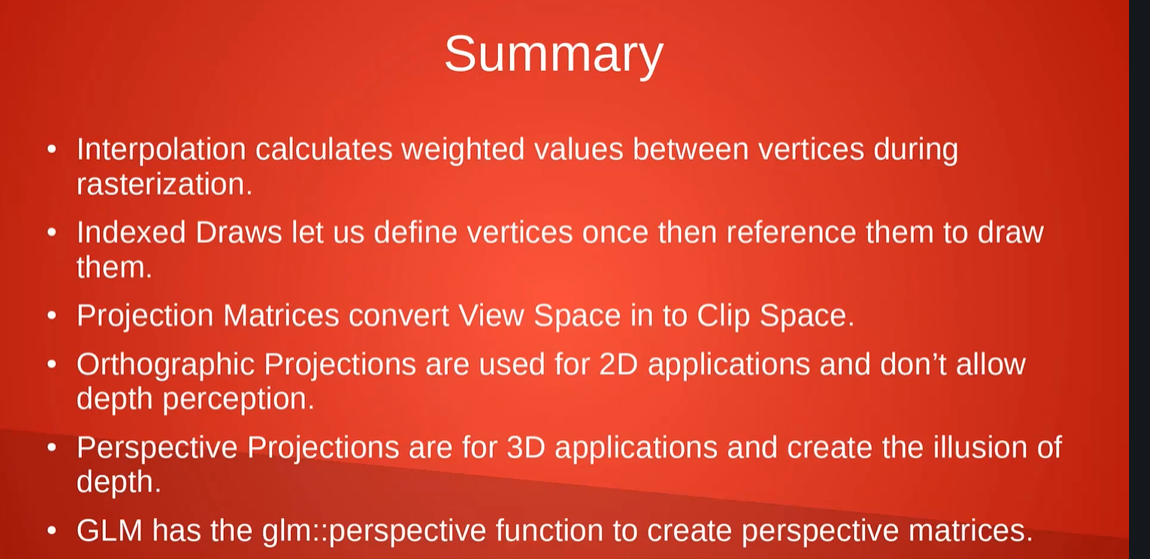
**Uniform Variable**

****

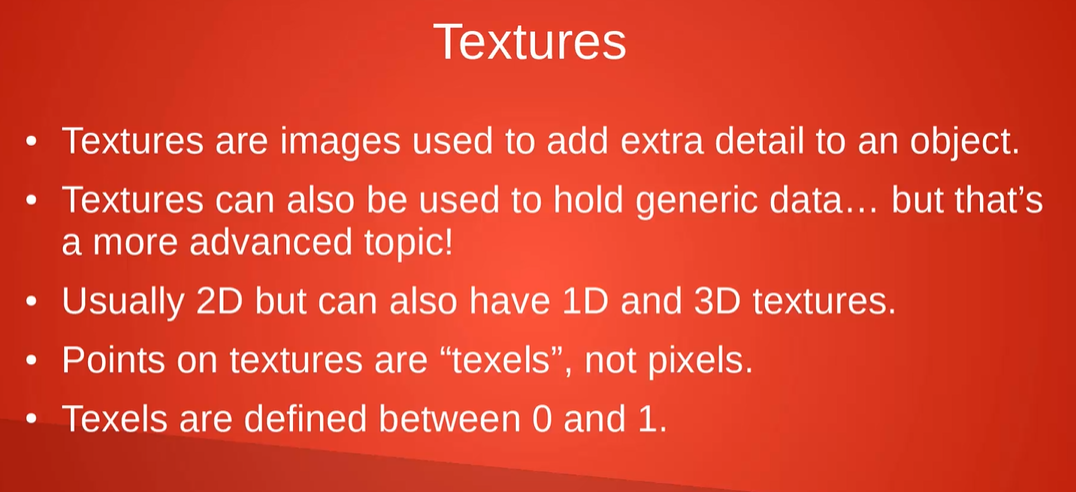
****

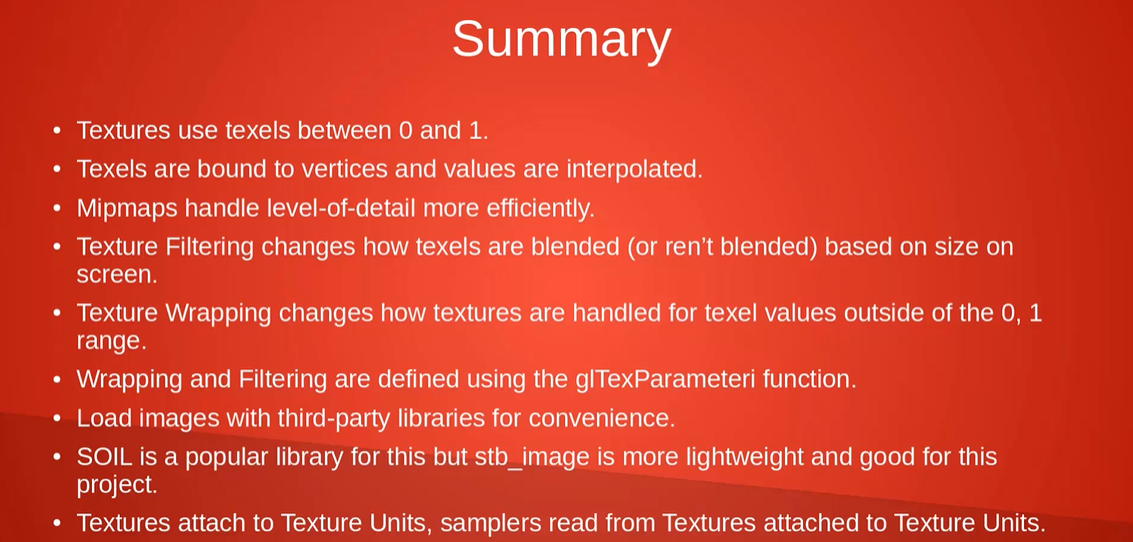
****

**Interpolation/Index Draws and Projections**

****

**Textures and Image Loading**

****

****