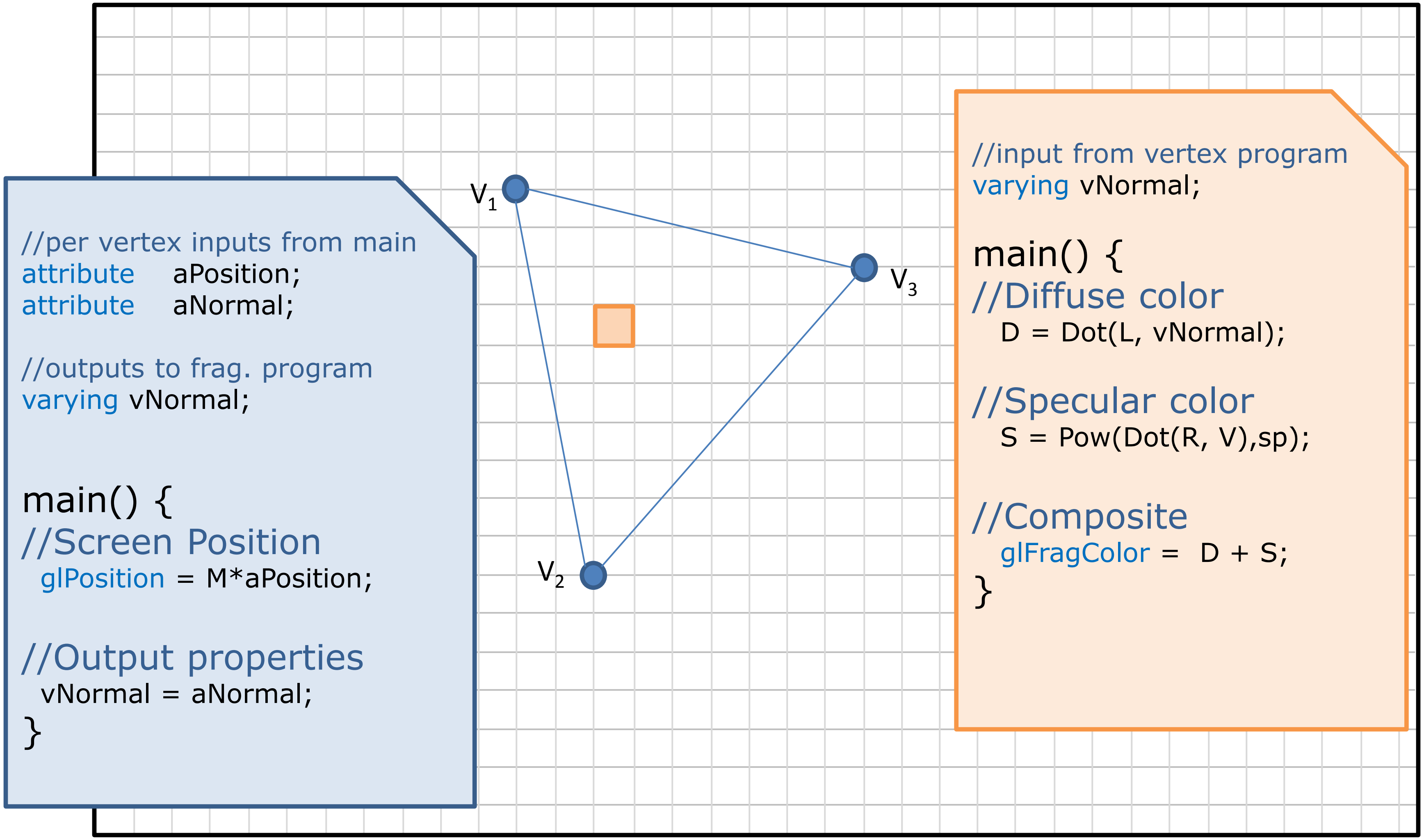
GLSL Introduction

Fu-Chung Huang

Thanks for materials from many other people

Programmable Shaders



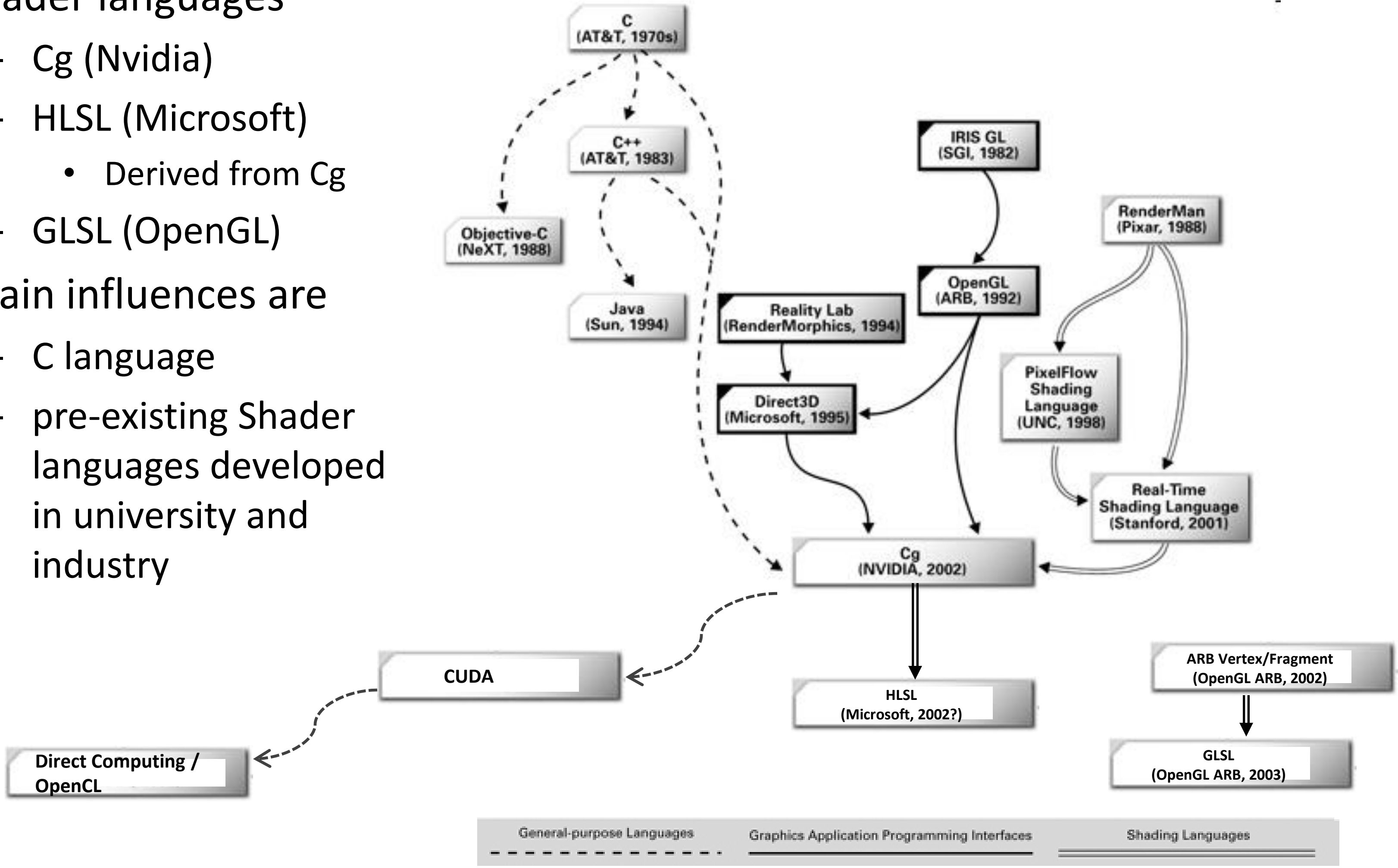
```
//per vertex inputs from main
attribute    aPosition;
attribute    aNormal;

//outputs to frag. program
varying vNormal;


main() {
//Screen Position
  glPosition = M*aPosition;

//Output properties
  vNormal = aNormal;
}
```

```
//input from vertex program
varying vNormal;

main() {
//Diffuse color
   D = Dot(L, vNormal);

//Specular color
   S = Pow(Dot(R, V),sp);

//Composite
   glFragColor =  D + S;
}
```

Shader Languages

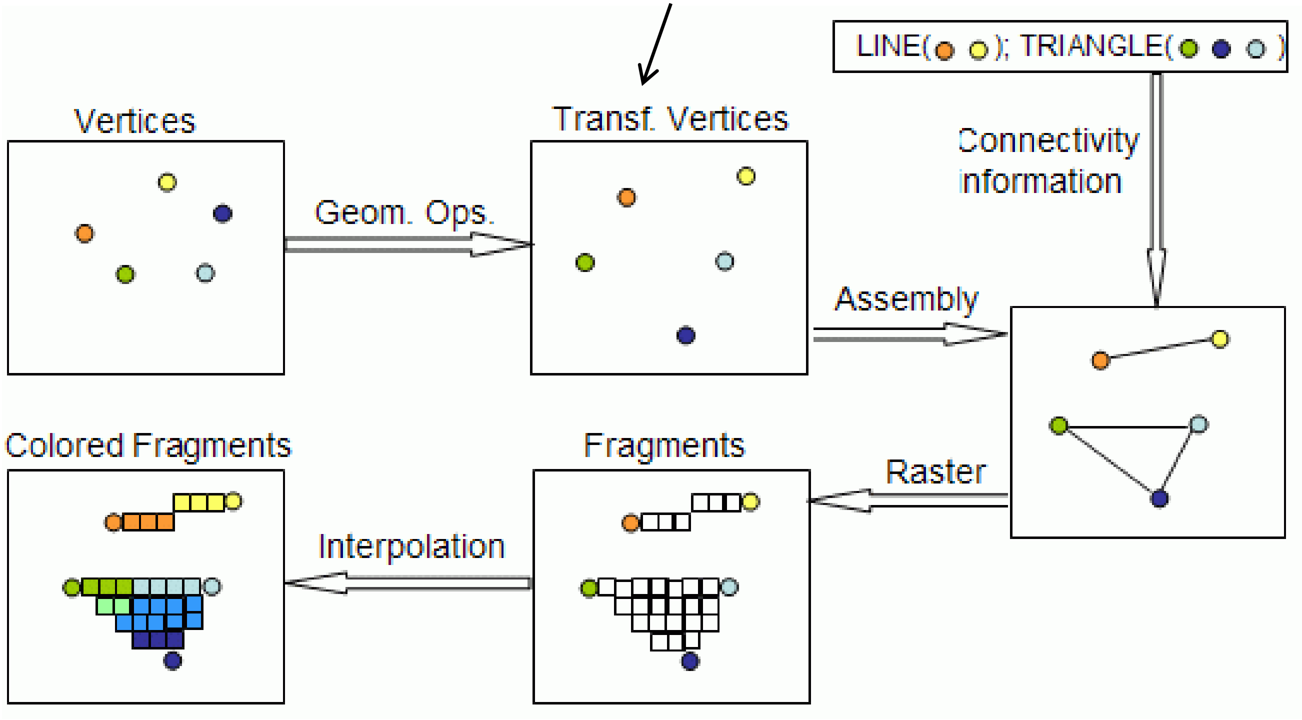
- Currently 3 major shader languages
  - Cg (Nvidia)
  - HLSL (Microsoft)
    - Derived from Cg
  - GLSL (OpenGL)
- Main influences are
  - C language
  - pre-existing Shader languages developed in university and industry

Source: http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter01.html (Modified with information on HLSL and GLSL)


C (AT&T, 1970s)
C++ (AT&T, 1983)
IRIS GL (SGI, 1982)
RenderMan (Pixar, 1988)
Objective-C (NeXT, 1988)
OpenGL (ARB, 1992)
Java (Sun, 1994)
Reality Lab (RenderMorphics, 1994)
PixelFlow Shading Language (UNC, 1998)
Direct3D (Microsoft, 1995)
Real-Time Shading Language (Stanford, 2001)
Cg (NVIDIA, 2002)
CUDA
ARB Vertex/Fragment (OpenGL ARB, 2002)
HLSL (Microsoft, 2002?)
GLSL (OpenGL ARB, 2003)
Direct Computing / OpenCL
General-purpose Languages
Graphics Application Programming Interfaces
Shading Languages

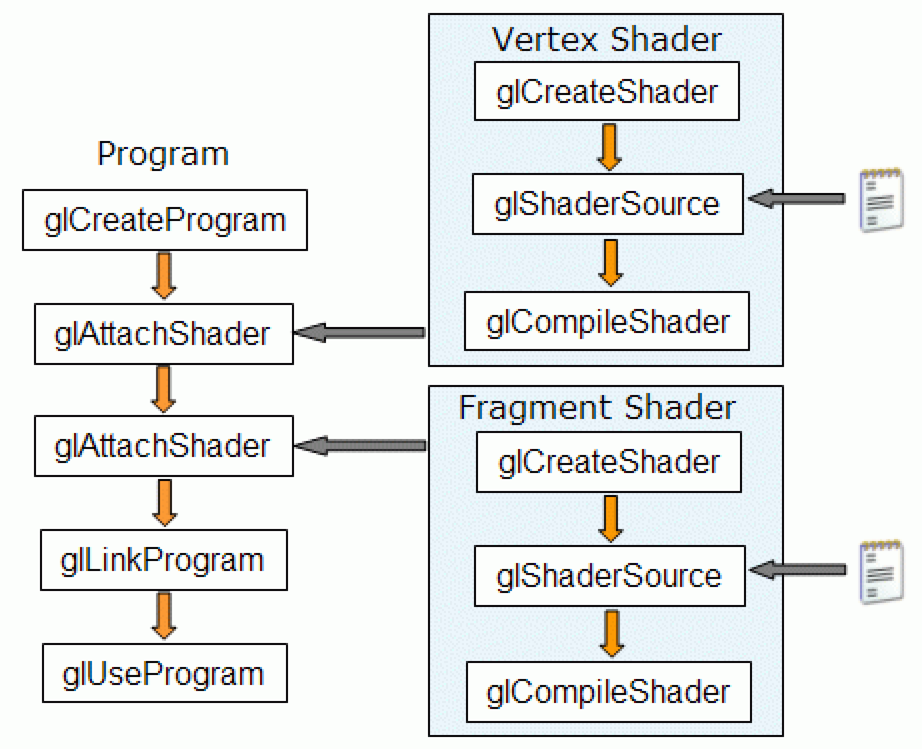
Fixed Functionality

Vertex Shader

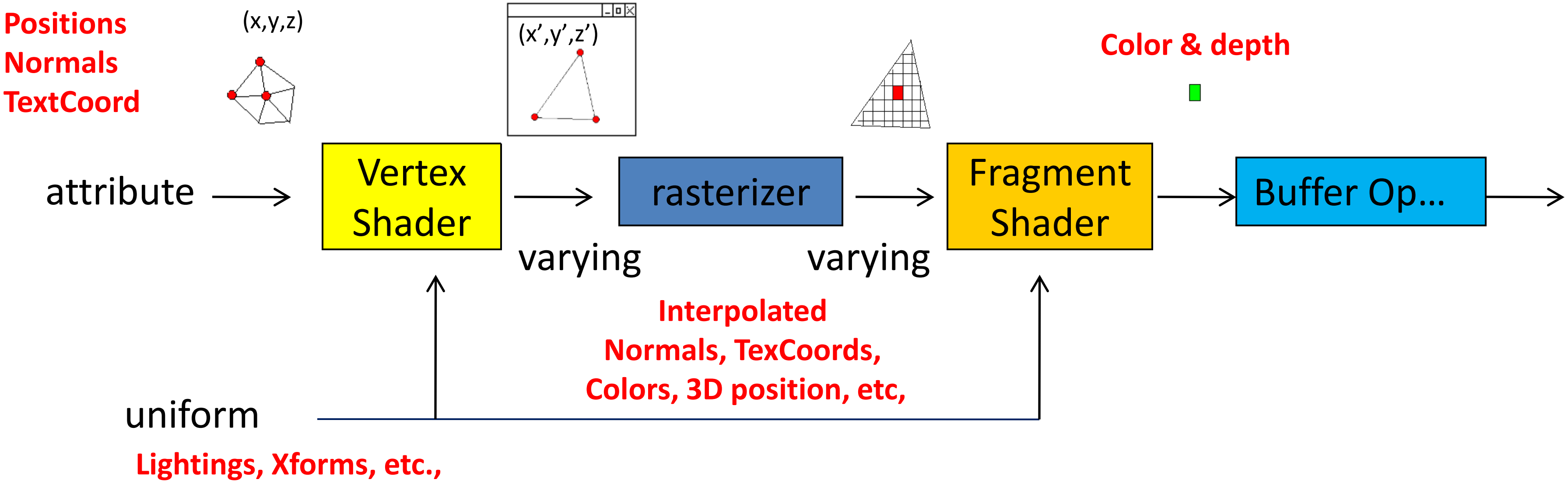


Frag. Shader

Shader Initialization



Qualifiers in pipeline



Positions
Normals
TextCoord

Color & depth

attribute → Vertex Shader → rasterizer → Fragment Shader → Buffer Op… →

varying

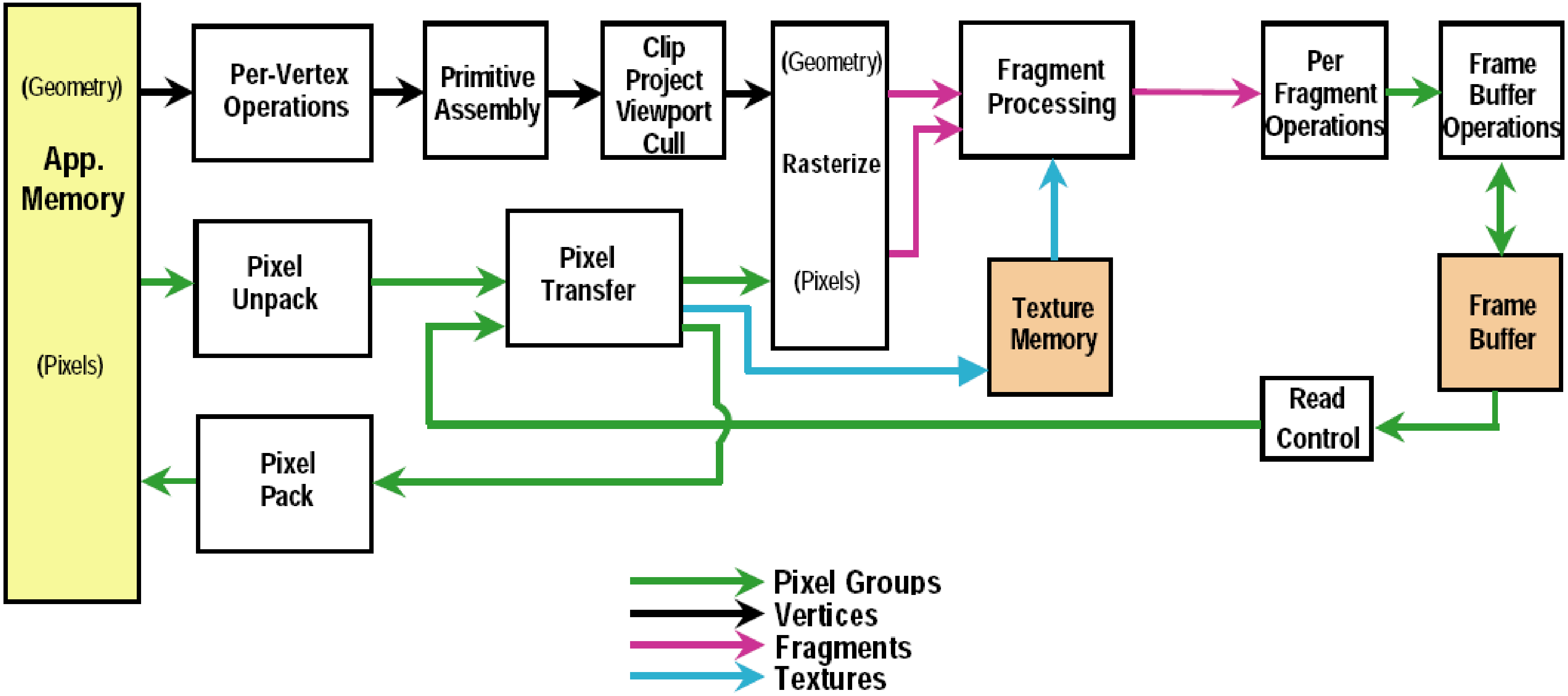
varying

Interpolated
Normals, TexCoords,
Colors, 3D position, etc,

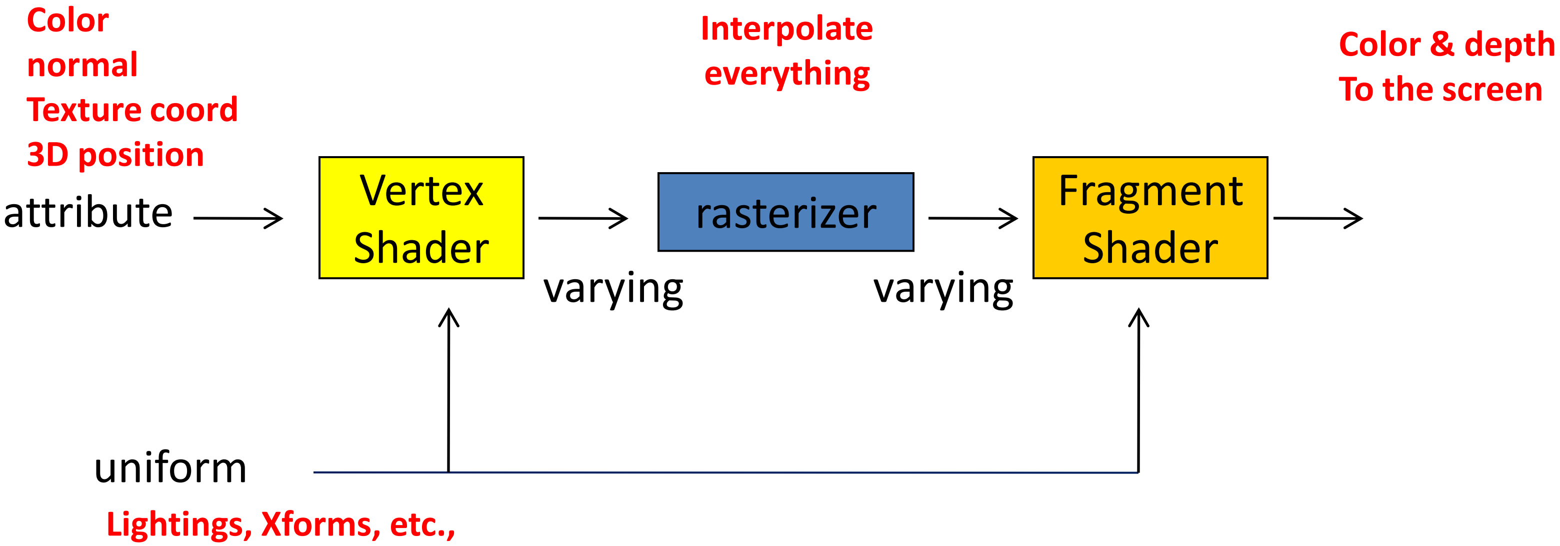
uniform

Lightings, Xforms, etc.,

Really Complicated Pipeline



Simplified Data Flow



Color
normal
Texture coord
3D position

attribute

Vertex Shader

varying

Interpolate everything

rasterizer

varying

Fragment Shader

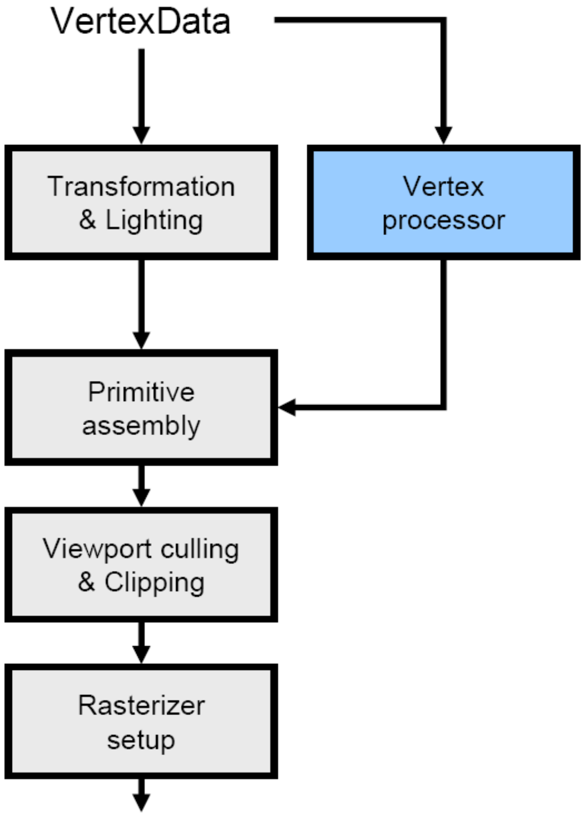
Color & depth
To the screen

uniform

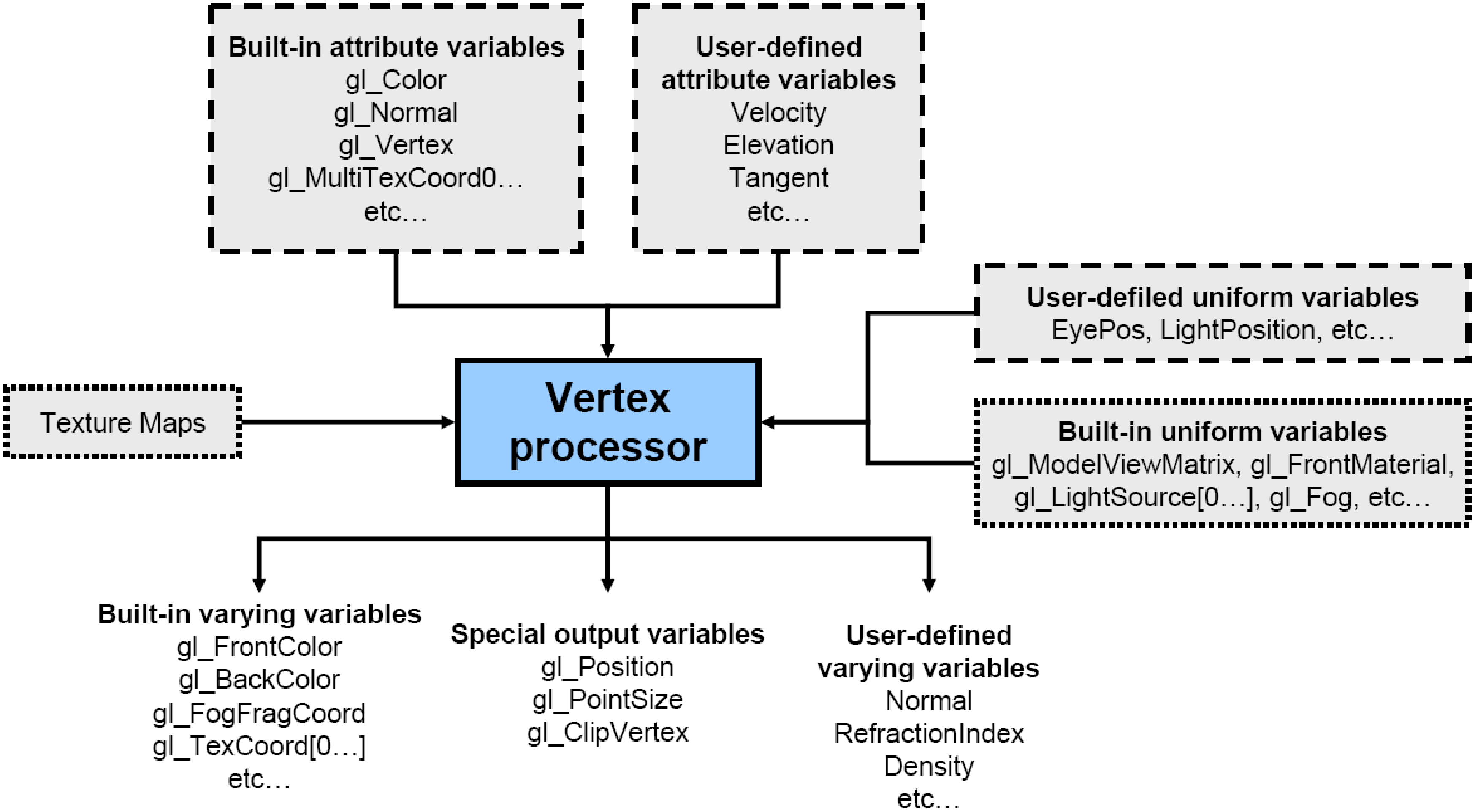
Lightings, Xforms, etc.,

Vertex Shader

- Vertex Xform
- Normal Xform
- Text Coord
- Per-vertex lighting



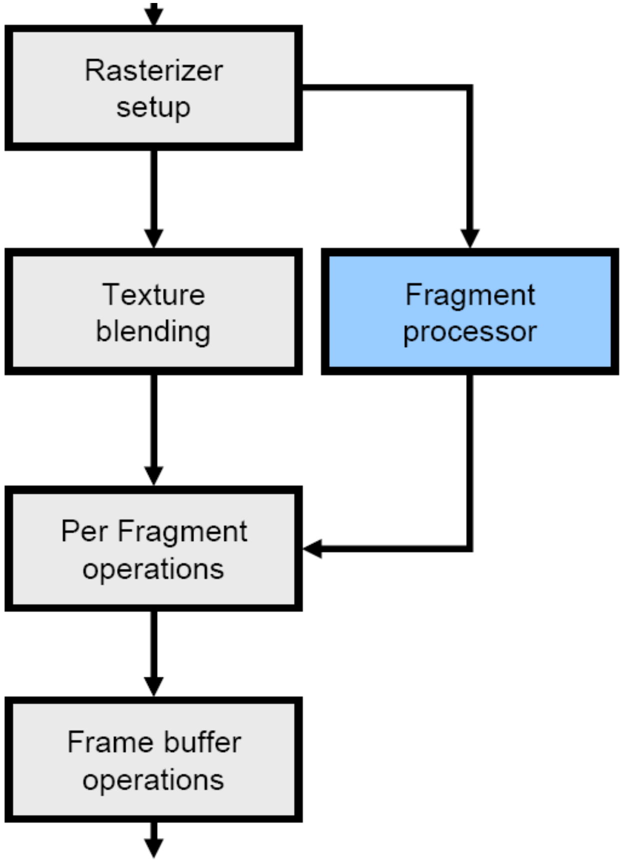
Vertex Shader



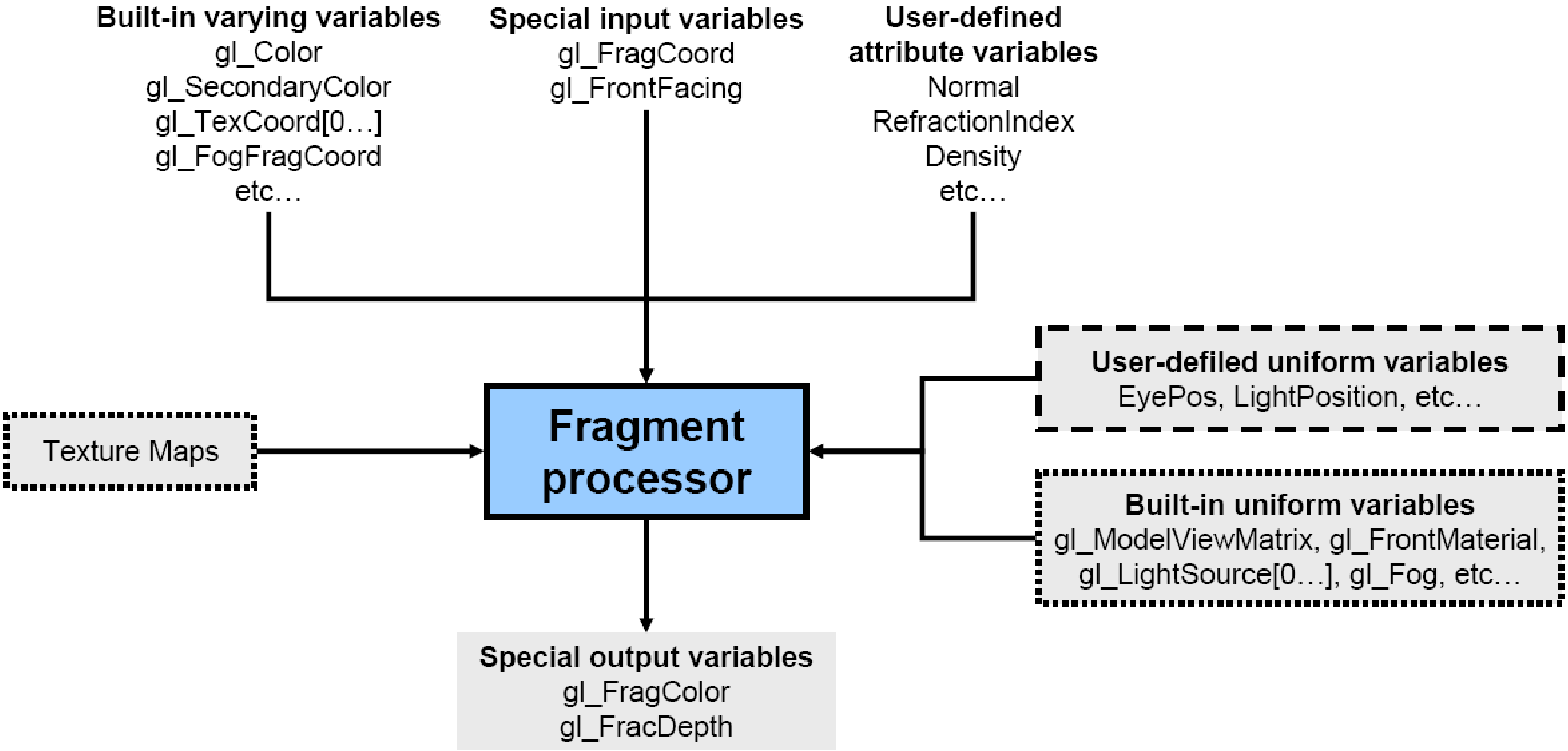
Built-in attribute variables
gl_Color
gl_Normal
gl_Vertex
gl_MultiTexCoord0…
etc…
User-defined attribute variables
Velocity
Elevation
Tangent
etc…
User-defiled uniform variables
EyePos, LightPosition, etc…
Texture Maps
Vertex processor
Built-in uniform variables
gl_ModelViewMatrix, gl_FrontMaterial, gl_LightSource[0…], gl_Fog, etc…
Built-in varying variables
gl_FrontColor
gl_BackColor
gl_FogFragCoord
gl_TexCoord[0…]
etc…
Special output variables
gl_Position
gl_PointSize
gl_ClipVertex
User-defined varying variables
Normal
RefractionIndex
Density
etc…

Fragment (pixel) Shader

- Interpolated
- Texture access
- Applications
  - Texture
  - Fog
  - Color sum



Fragment Shader



Built-in varying variables
gl_Color
gl_SecondaryColor
gl_TexCoord[0…]
gl_FogFragCoord
etc…

Special input variables
gl_FragCoord
gl_FrontFacing

User-defined attribute variables
Normal
RefractionIndex
Density
etc…

Texture Maps

Fragment processor

User-defiled uniform variables
EyePos, LightPosition, etc…

Built-in uniform variables
gl_ModelViewMatrix, gl_FrontMaterial, gl_LightSource[0…], gl_Fog, etc…

Special output variables
gl_FragColor
gl_FracDepth

GLSL Language Definition

- Data Type Description

int	Integer
float	Floating-point
bool	Boolean (*true* or *false*).
vec2	Vector with two floats.
vec3	Vector with three floats.
vec4	Vector with four floats.
mat2	2x2 floating-point matrix.
mat3	3x3 floating-point matrix.
mat4	4x4 floating-point matrix.

Vector

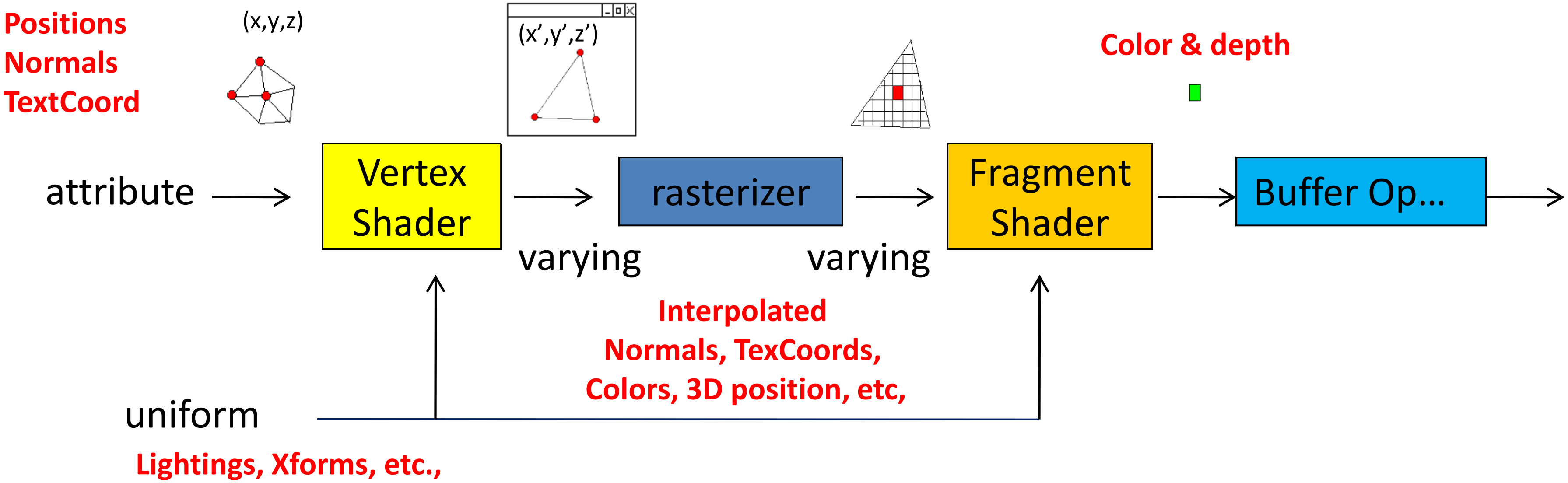
- Vector is like a class
- You can use following to access
  - .r .g .b .a
  - .x .y .z .w
  - .s .t .p .q
- Example:

```
vec4 color;
color.rgb      = vec3(1.0 , 1.0 , 0.0 );        color.a = 0.5
color          = vec4(1.0 , 1.0 , 0.0 , 0.5);
color.xy       = vec2(1.0 , 1.0);
color.zw       = vec2(0.0 , 0.5);
```

GLSL Variable Qualifiers

- Qualifiers give a special meaning to the variable. In GLSL the following qualifiers are available:
  - **const** - the declaration is of a compile time constant
  - **uniform** – (used both in vertex/fragment shaders, read-only in both) global variables that may change per primitive (may not be set inside glBegin,/glEnd)
  - **varying** - used for interpolated data between a vertex shader and a fragment shader. Available for writing in the vertex shader, and read-only in a fragment shader.
  - **attribute** – (only used in vertex shaders, and read-only in shader) global variables that may change per vertex, that are passed from the OpenGL application to vertex shaders.

Qualifiers in pipeline



Positions
Normals
TextCoord

(x,y,z)

attribute

Vertex Shader

(x',y',z')

varying

rasterizer

varying

Fragment Shader

Color & depth

Buffer Op…

Interpolated
Normals, TexCoords,
Colors, 3D position, etc,

uniform

Lightings, Xforms, etc.,

Vertex Shader Code Example

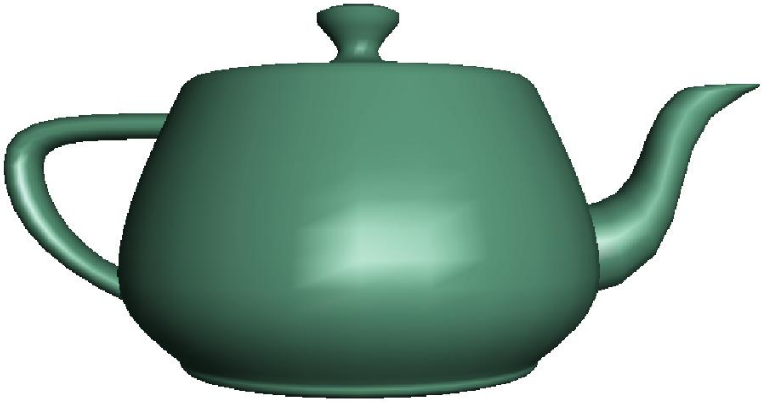
```
uniform   mat4 uMVP, uMV, uN;
uniform   vec4 uEye, uLight, uLightColor, uKd, uKs;
attribute vec4 aPos, aNorm;        //input
varying   vec4 vPos, vNorm;        //output

void main()
{
     // get the screen coordinate for rasterizer
     // HW2 use gl_ModelViewMatrix and gl_ProjectionMatrix, gl_Vertex
    gl_Position = vec3(uMVP * aPos);


     // pass output to rasterizer, interpolate, and as input to frag. shader
    vNorm = aNorm;
    vPos = aPos;
}
```

Fragment Shader Code Example

```
uniform mat4 uMVP, uMV, uN;
uniform vec4 uEye, uLight, uLightColor, uKd, uKs;
//not using attribute
varying vec4 vPos, vNorm;                  //inpute from VS

void main (void)
{
    vec3 V = vec3(uMV*vPos);               //why just ModelView?
    vec3 N = normalize(vec3(uN*vNorm));    //uN = uMV^-T
    vec3 L = normalize(vec3(uLight));      //depends on spec.

    float lambertTerm = max(dot(N,L),0);   //diffuse component
    vec4 diffuse = uLightColor * uKd * lambertTerm;

    //Finally specular term, HW2 requires Blinn-Phing
    vec3 E = normalize(-V);                //why -V?
    vec3 R = reflect(-L, N);
    float specularTerm = pow( max(dot(R, E), 0.0),   shininess );
    vec4 specular = uLightColor * uKs * specularTerm;

    gl_FragColor = diffuse + specular;
}
```

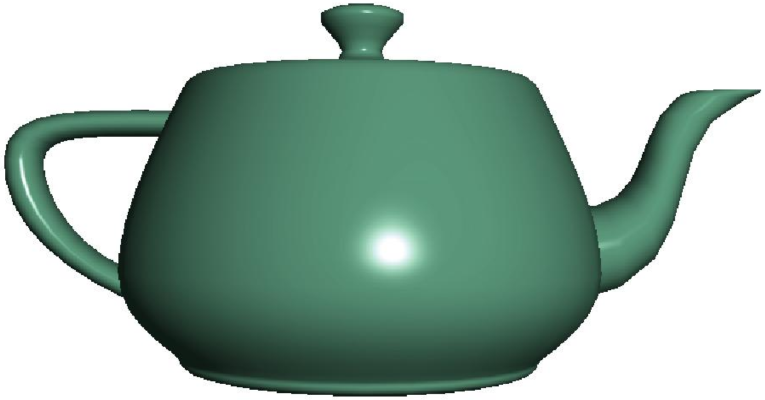
Vertex vs. Fragment Shader

Smooth Shading



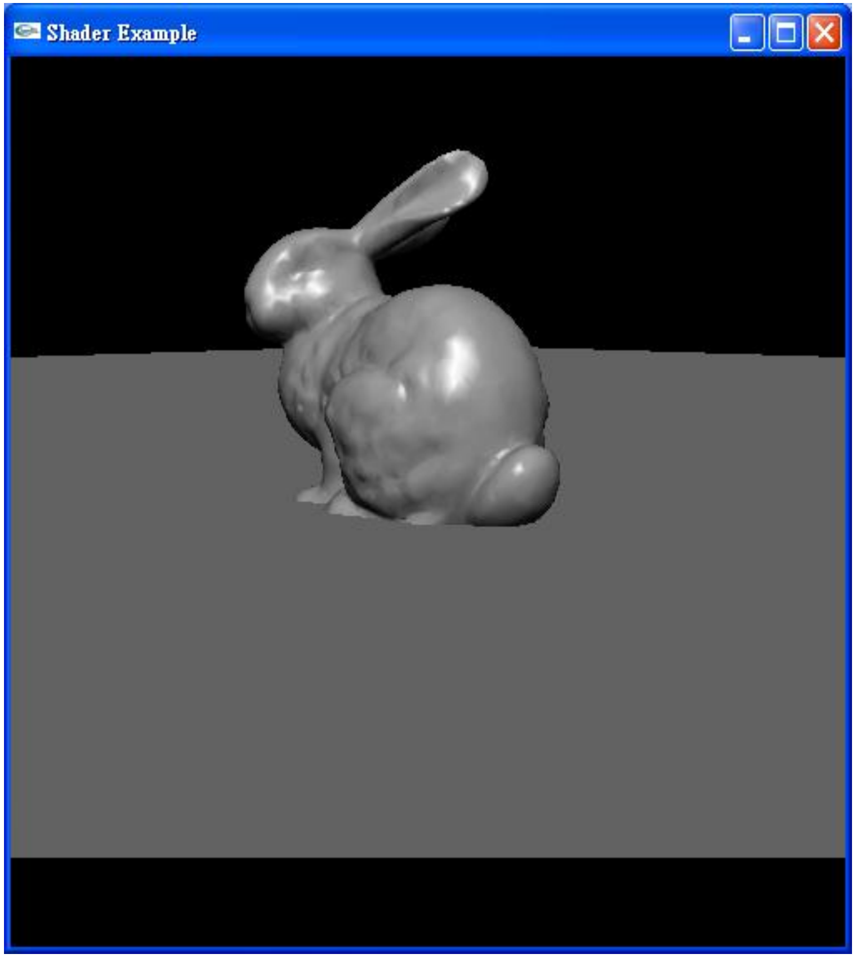
per vertex lighting

Phong Shading

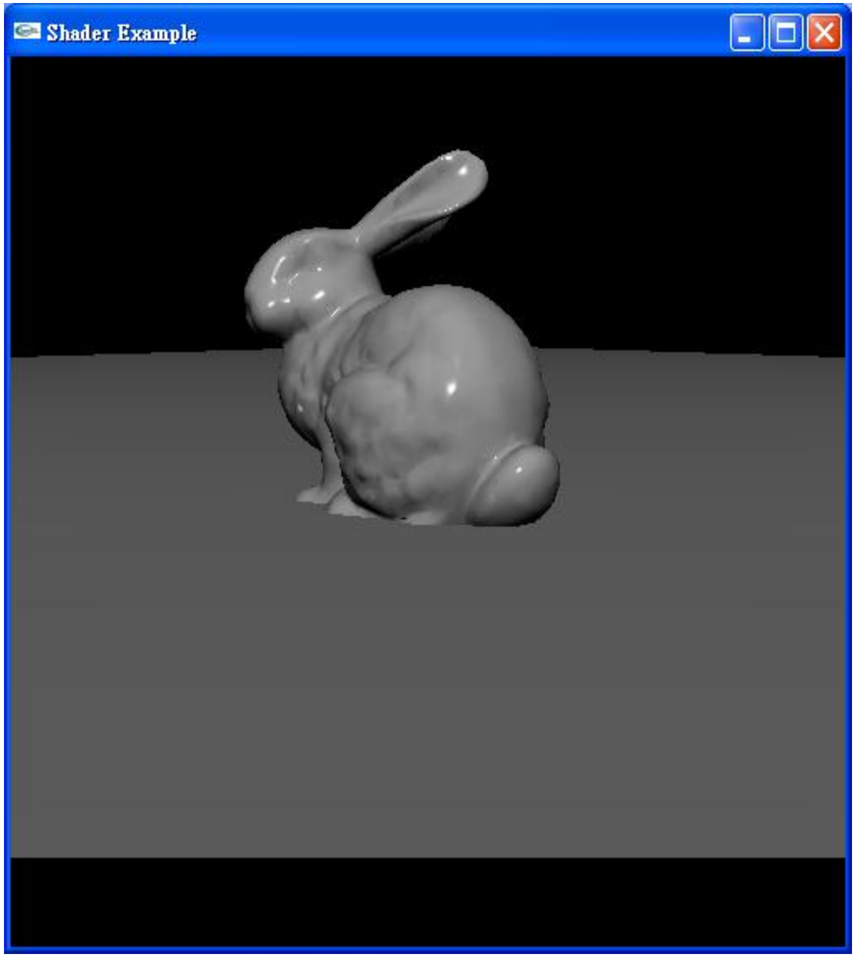


per fragment lighting

Result



OpenGL Gouraud Shading



GLSL Phong Shading

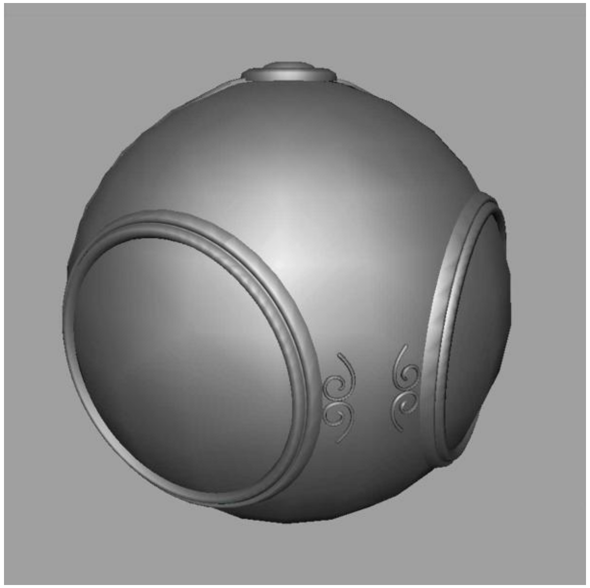
GLSL Statements

- Control Flow Statements: pretty much the same as in C.
- HIGHLY HARDWARE DEPENDENT!!

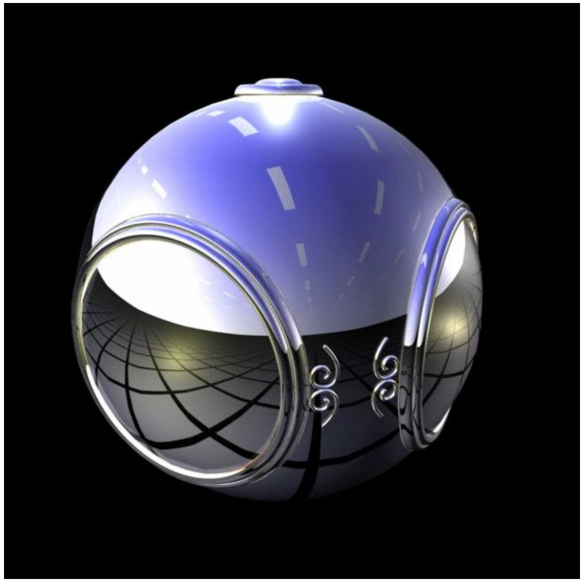
```
if (bool expression)
    …
else
   …

for (initialization; bool expression; loop expression)
   …

while (bool expression)
  …

do
 …
while (bool expression)
```

Note: only “if” are available on most current hardware

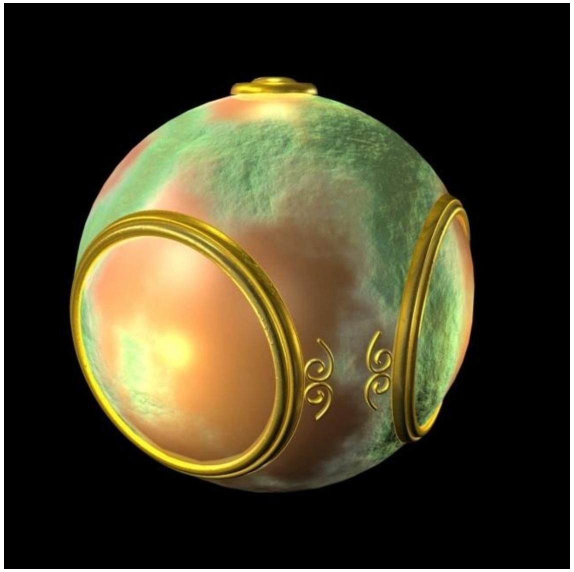
Fragment Shader Applications



smooth shading



environment mapping



bump mapping

Bump Mapping

- Perturb normal for each fragment
- Store perturbation as textures