GPU Shading

- Announcements
- GPU Shading Overview
- Example 1: Shadow Maps
- Example 2: Algebraic Surfaces

GPU Shading

- Announcements
- GPU Shading Overview
- Example 1: Shadow Maps
- Example 2: Algebraic Surfaces

Announcements

- Homework 2 Due
- Project 3 Released
- Project 3 Vote
- Midterm Exam Next Week!

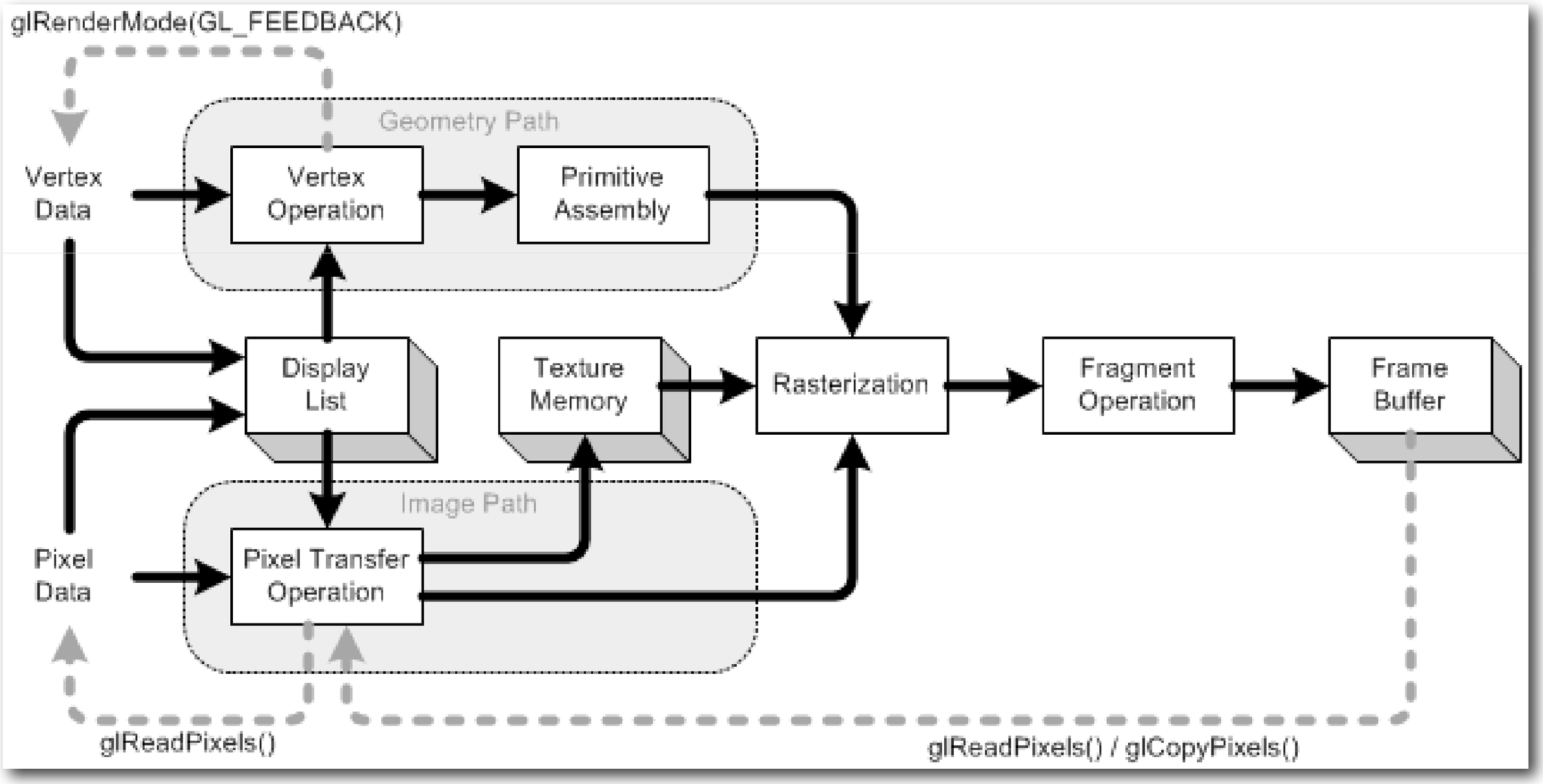
GPU Shading

- Announcements
- GPU Shading Overview
- Example 1: Shadow Maps
- Example 2: Algebraic Surfaces

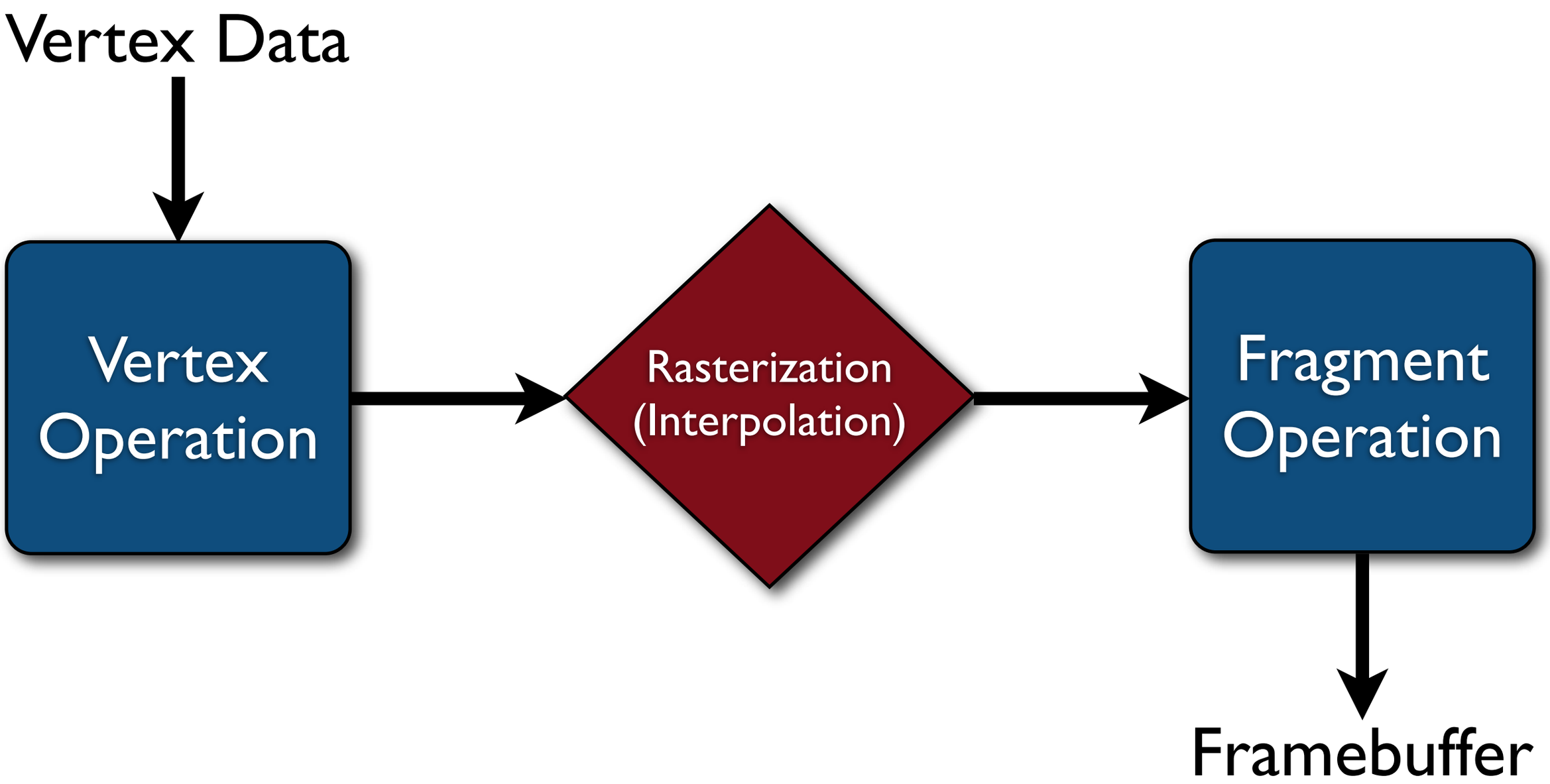
GPU Shading

- Announcements
- GPU Shading Overview
- Example 1: Shadow Maps
- Example 2: Algebraic Surfaces

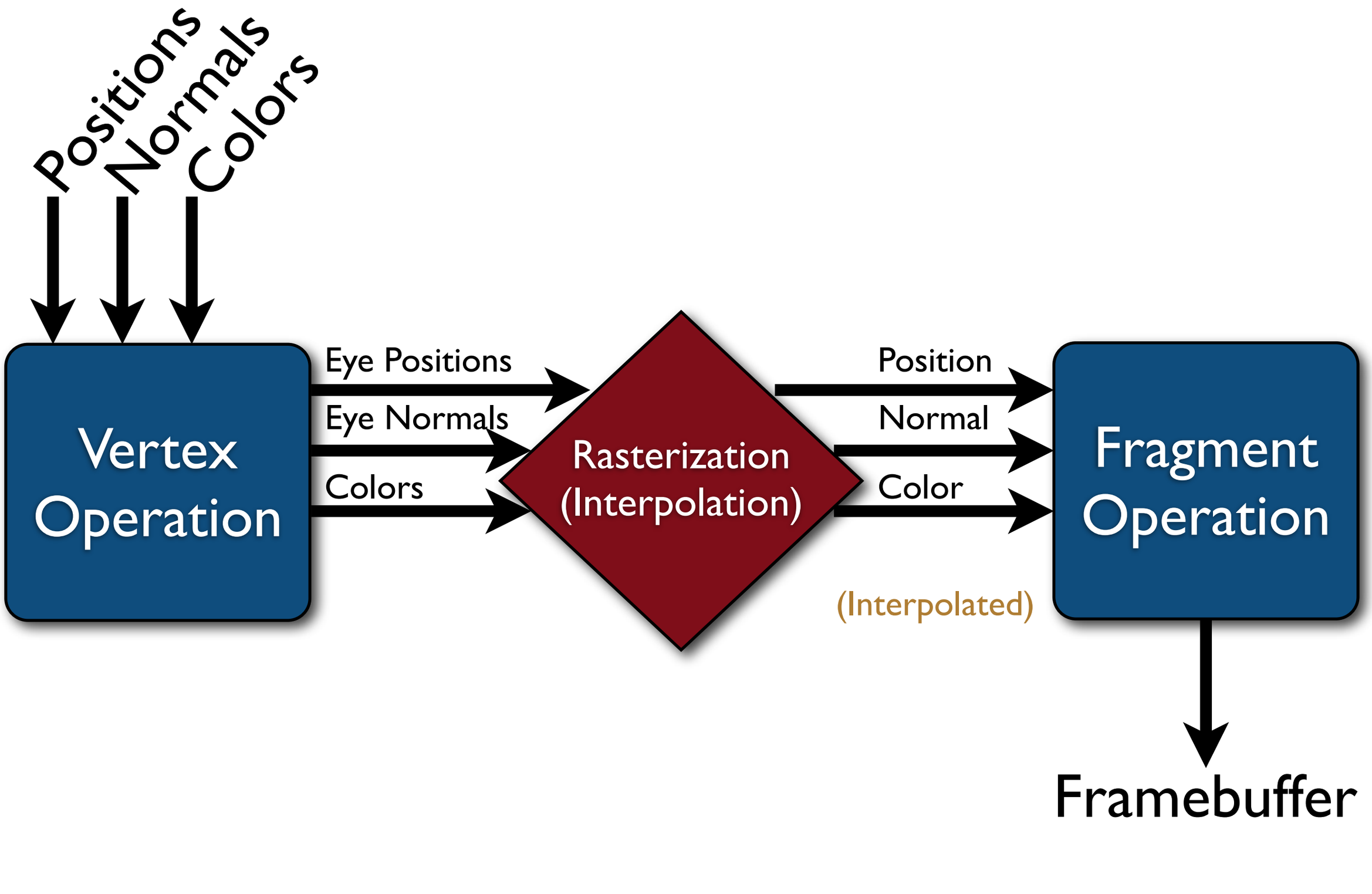
OpenGL Pipeline



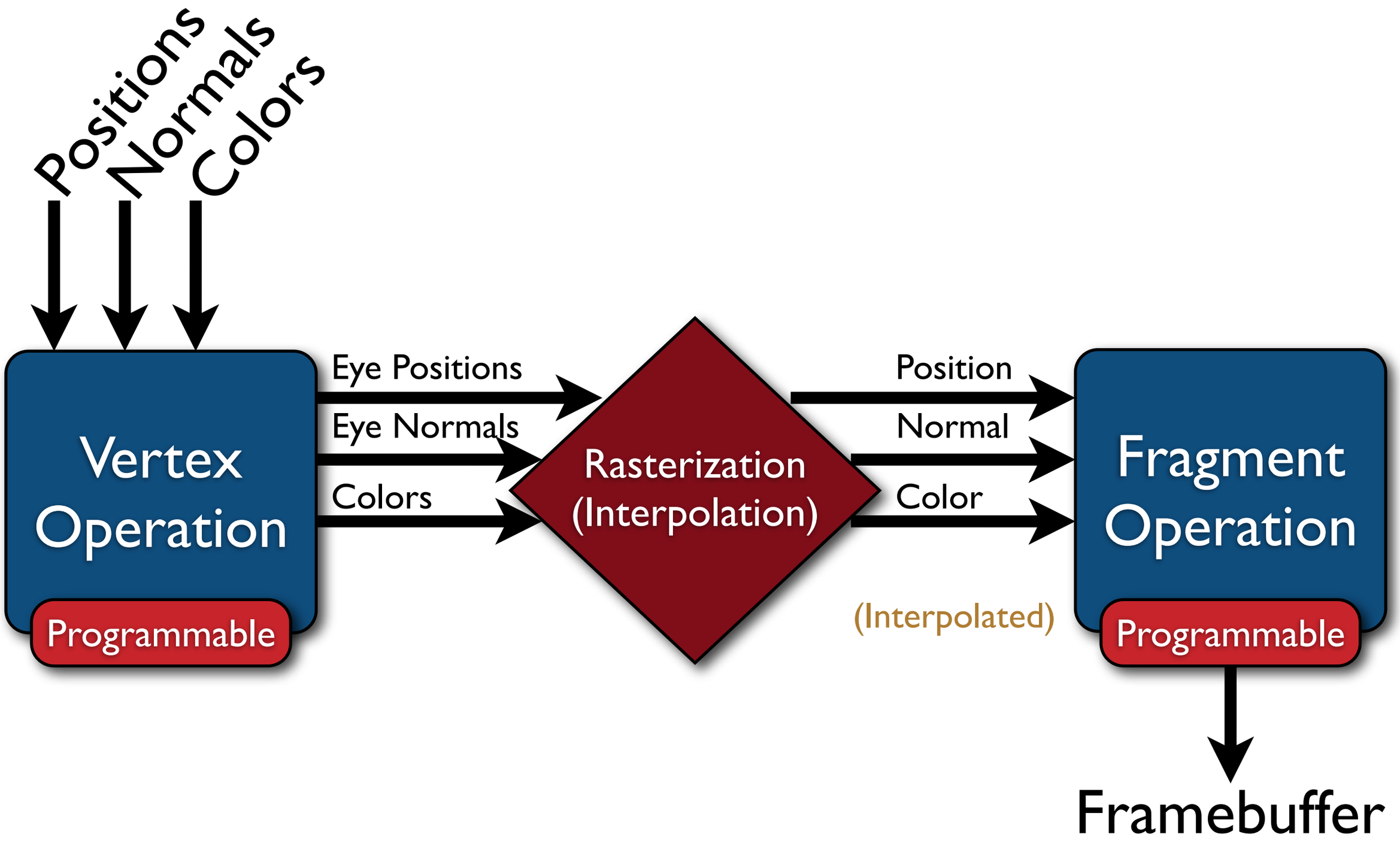
Simplified OpenGL Pipeline



More Detail


Positions
Normals
Colors
Vertex Operation
Eye Positions
Eye Normals
Colors
Rasterization (Interpolation)
Position
Normal
Color
(Interpolated)
Fragment Operation
Framebuffer

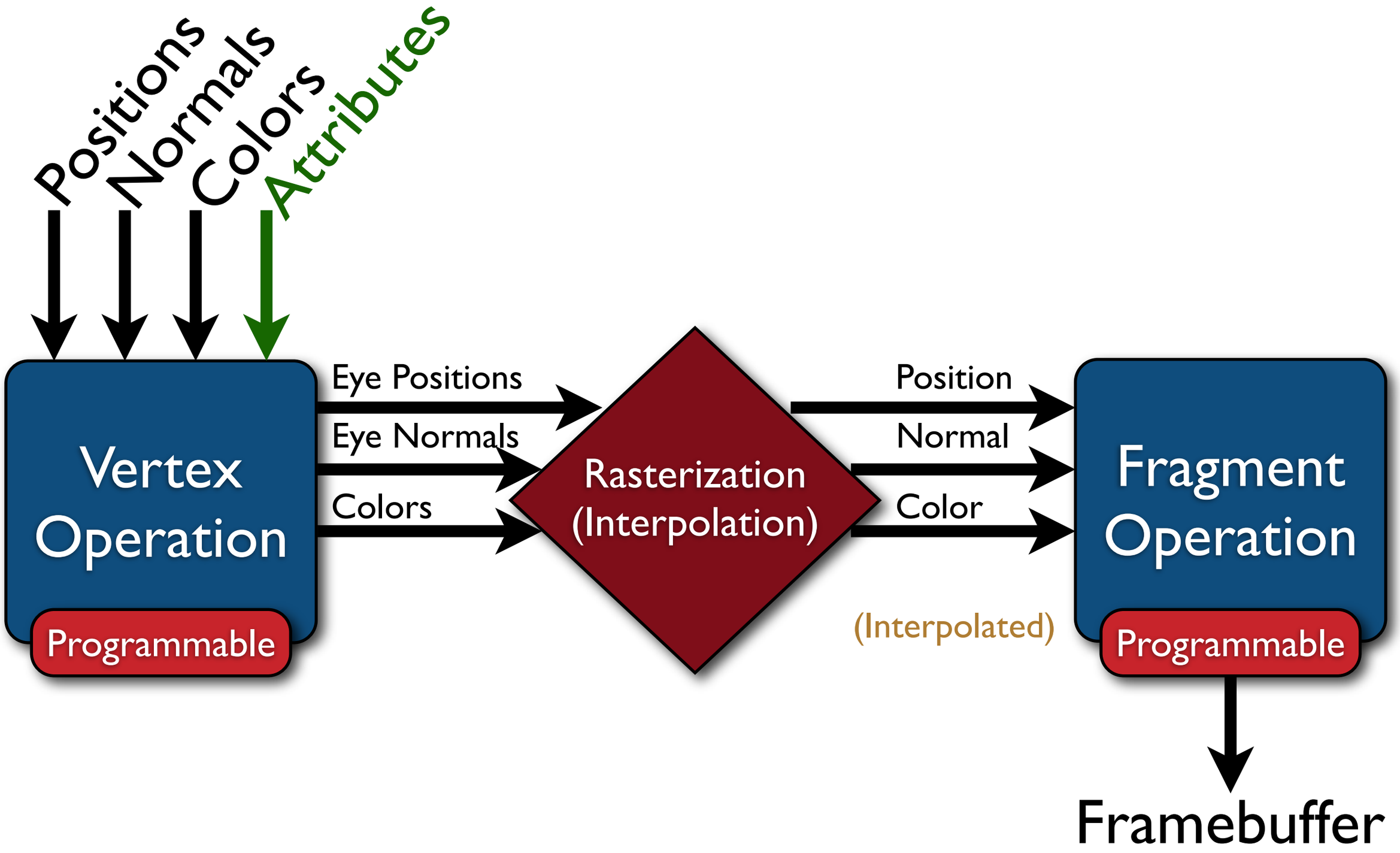
Programmability



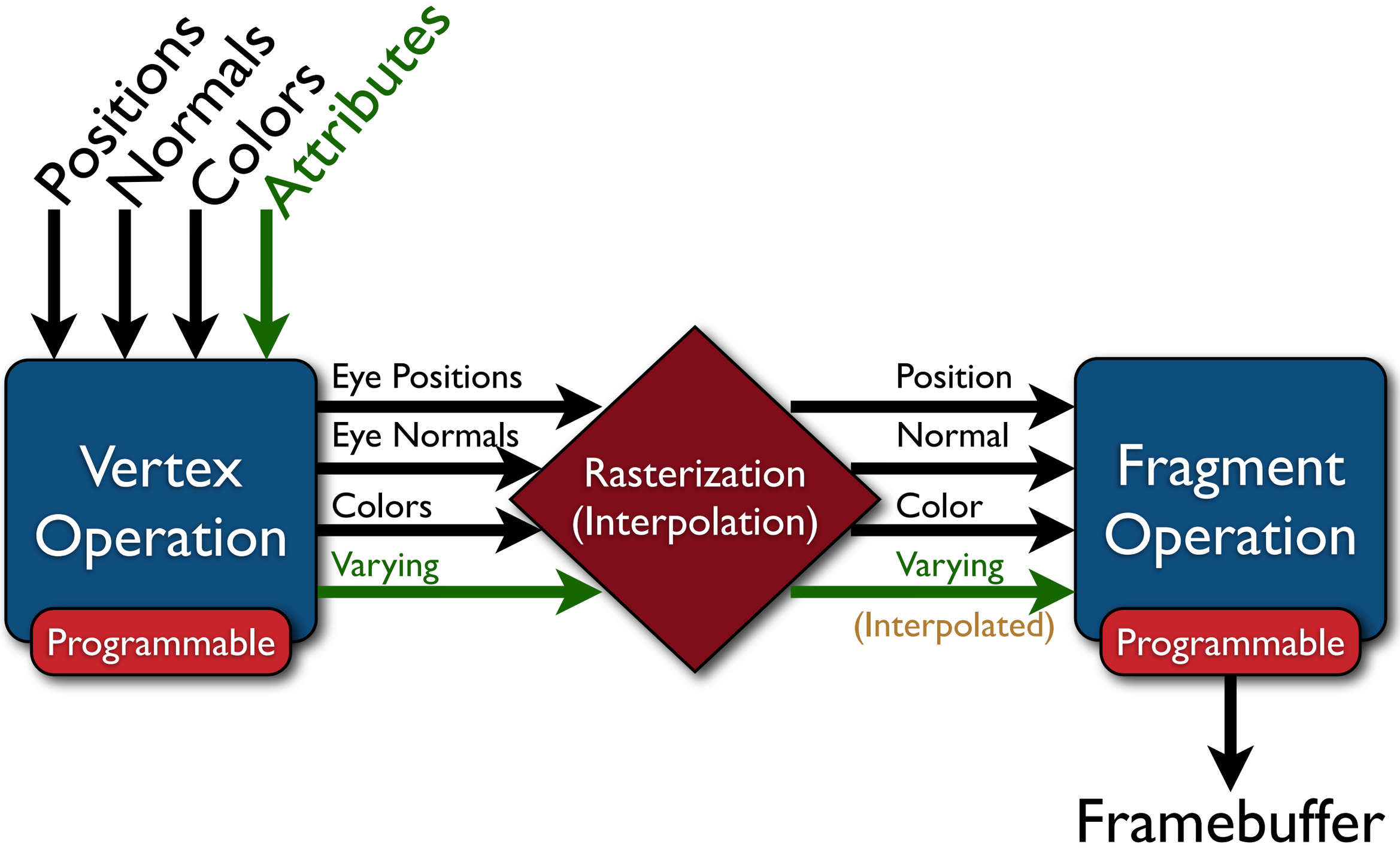
Variable Types

- **const** - The declaration is of a compile time constant
- **attribute** - Global variables that may change per vertex, that are passed from the OpenGL application to vertex shaders. This qualifier can only be used in vertex shaders. For the shader this is a read-only variable.
- **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.
- **uniform** - Global variables that may change per primitive (may not be set inside glBegin,/glEnd), that are passed from the OpenGL application to the shaders. This qualifier can be used in both vertex and fragment shaders. For the shaders this is a read-only variable.

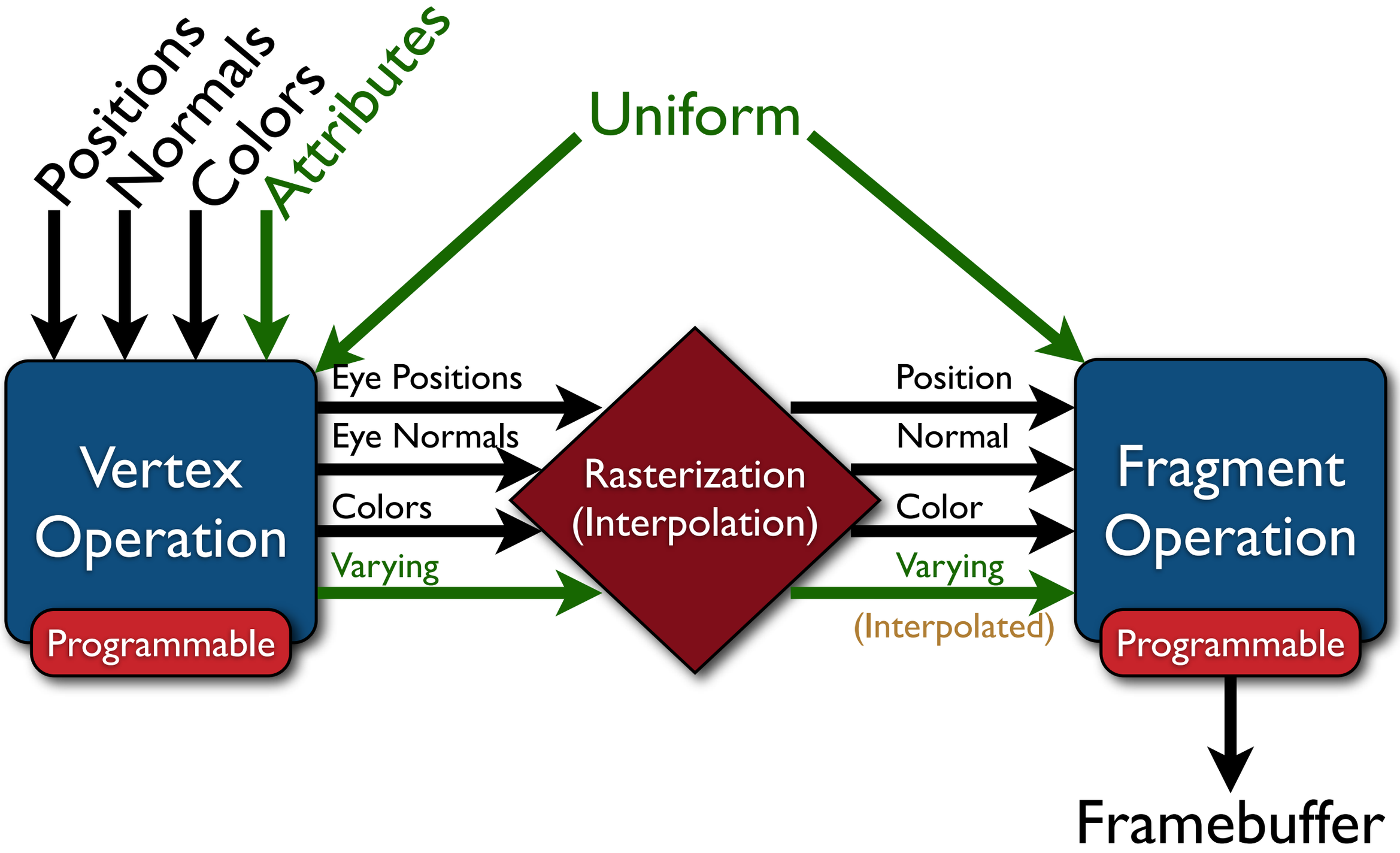
Attribute Variables



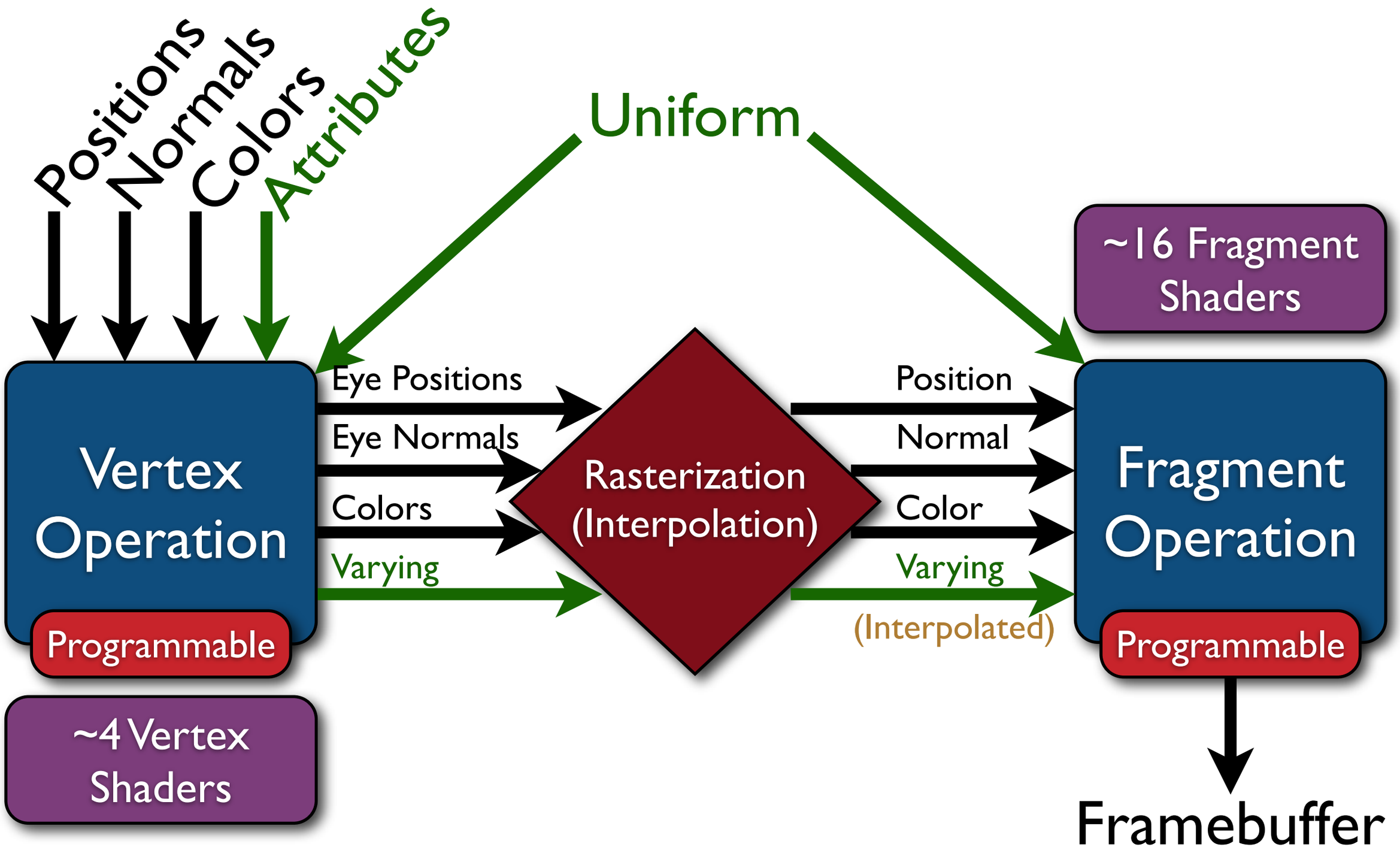
Varying Variables



Uniform Variables



Why So Fast?


Positions
Normals
Colors
Attributes
Uniform
~16 Fragment Shaders
Vertex Operation
Programmable
~4 Vertex Shaders
Eye Positions
Eye Normals
Colors
Varying
Rasterization (Interpolation)
Position
Normal
Color
Varying
(Interpolated)
Fragment Operation
Programmable
Framebuffer

Types and Functions

OpenGL® Shading Language (GLSL) Quick Reference Guide

Describes GLSL version 1.10, as included in OpenGL v2.0, and specified by "The OpenGL® Shading Language", version 1.10.59. Section and page numbers refer to that version of the spec.

DATA TYPES (4.1 p16)

float, vec2, vec3, vec4
int, ivec2, ivec3, ivec4
bool, bvec2, bvec3, bvec4
mat2, mat3, mat4
void
sampler1D, sampler2D, sampler3D
samplerCube
sampler1DShadow, sampler2DShadow

DATA TYPE QUALIFIERS (4.3 p22)

global variable declarations:

uniform	input to Vertex and Fragment shader from OpenGL or application (READ-ONLY)
attribute	input per-vertex to Vertex shader from OpenGL or application (READ-ONLY)
varying	output from Vertex shader (READ/WRITE), interpolated, then input to Fragment shader (READ-ONLY)
const	compile-time constant (READ-ONLY)

function parameters:

in	value initialized on entry, not copied on return (default)
out	copied out on return, but not initialized
inout	value initialized on entry, and copied out on return
const	constant function input

VECTOR COMPONENTS (5.5 p 30)

component names may not be mixed across sets

x, y, z, w
r, g, b, a
s, t, p, q

PREPROCESSOR (3.3 p9)

#
#define
#undef
#if
#ifdef
#ifndef
#else
#elif
#endif
#error
#pragma
#line

__LINE__
__FILE__
__VERSION__

GLSL version declaration and extensions protocol:
#version
default is "#version 110" (3.3 p11)
#extension {*name* | all} : {require | enable | warn | disable}
default is "#extension all : disable" (3.3 p11)

BUILT-IN FUNCTIONS

Key:
vec = vec2 | vec3 | vec4
mat = mat2 | mat3 | mat4
ivec = ivec2 | ivec3 | ivec4
bvec = bvec2 | bvec3 | bvec4
genType = float | vec2 | vec3 | vec4

Angle and Trigonometry Functions (8.1 p51)
genType sin(genType)
genType cos(genType)
genType tan(genType)

genType asin(genType)
genType acos(genType)
genType atan(genType, genType)
genType atan(genType)

genType radians(genType)
genType degrees(genType)

Exponential Functions (8.2 p52)
genType pow(genType, genType)
genType exp(genType)
genType log(genType)
genType exp2(genType)
genType log2(genType)
genType sqrt(genType)
genType inversesqrt(genType)

Common Functions (8.3 p52)
genType abs(genType)
genType ceil(genType)
genType clamp(genType, genType, genType)
genType clamp(genType, float, float)
genType floor(genType)
genType fract(genType)
genType max(genType, genType)
genType max(genType, float)
genType min(genType, genType)
genType min(genType, float)
genType mix(genType, genType, genType)
genType mix(genType, genType, float)
genType mod(genType, genType)
genType mod(genType, float)
genType sign(genType)
genType smoothstep(genType, genType, genType)
genType smoothstep(float, float, genType)
genType step(genType, genType)
genType step(float, genType)

Geometric Functions (8.4 p54)

vec4	ftransform()	*Vertex ONLY*
vec3	cross(vec3, vec3)	
float	distance(genType, genType)	
float	dot(genType, genType)	
genType	faceforward(genType *V*, genType *I*, genType *N*)	
float	length(genType)	
genType	normalize(genType)	
genType	reflect(genType *I*, genType *N*)	
genType	refract(genType *I*, genType *N*, float *eta*)	

Fragment Processing Functions (8.8 p58) *Fragment ONLY*
genType dFdx(genType)
genType dFdy(genType)
genType fwidth(genType)

Matrix Functions (8.5 p55)
mat matrixCompMult(mat, mat)

Vector Relational Functions (8.6 p55)
bool all(bvec)
bool any(bvec)
bvec equal(vec, vec)
bvec equal(ivec, ivec)
bvec equal(bvec, bvec)
bvec greaterThan(vec, vec)
bvec greaterThan(ivec, ivec)
bvec greaterThanEqual(vec, vec)
bvec greaterThanEqual(ivec, ivec)
bvec lessThan(vec, vec)
bvec lessThan(ivec, ivec)
bvec lessThanEqual(vec, vec)
bvec lessThanEqual(ivec, ivec)
bvec not(bvec)
bvec notEqual(vec, vec)
bvec notEqual(ivec, ivec)
bvec notEqual(bvec, bvec)

Texture Lookup Functions (8.7 p56)
Optional bias term is Fragment ONLY
vec4 texture1D(sampler1D, float [,float *bias*])
vec4 texture1DProj(sampler1D, vec2 [,float *bias*])
vec4 texture1DProj(sampler1D, vec4 [,float *bias*])

vec4 texture2D(sampler2D, vec2 [,float *bias*])
vec4 texture2DProj(sampler2D, vec3 [,float *bias*])
vec4 texture2DProj(sampler2D, vec4 [,float *bias*])

vec4 texture3D(sampler3D, vec3 [,float *bias*])
vec4 texture3DProj(sampler3D, vec4 [,float *bias*])

vec4 textureCube(samplerCube, vec3 [,float *bias*])

vec4 shadow1D(sampler1DShadow, vec3 [,float *bias*])
vec4 shadow2D(sampler2DShadow, vec3 [,float *bias*])
vec4 shadow1DProj(sampler1DShadow, vec4 [,float *bias*])
vec4 shadow2DProj(sampler2DShadow, vec4 [,float *bias*])

Texture Lookup Functions with LOD (8.7 p56)
Vertex ONLY; ensure GL_MAX_VERTEX_TEXTURE_IMAGE_UNITS > 0
vec4 texture1DLod(sampler1D, float, float *lod*)
vec4 texture1DProjLod(sampler1D, vec2, float *lod*)
vec4 texture1DProjLod(sampler1D, vec4, float *lod*)

vec4 texture2DLod(sampler2D, vec2, float *lod*)
vec4 texture2DProjLod(sampler2D, vec3, float *lod*)
vec4 texture2DProjLod(sampler2D, vec4, float *lod*)
vec4 texture3DProjLod(sampler3D, vec4, float *lod*)

vec4 textureCubeLod(samplerCube, vec3, float *lod*)

vec4 shadow1DLod(sampler1DShadow, vec3, float *lod*)
vec4 shadow2DLod(sampler2DShadow, vec3, float *lod*)
vec4 shadow1DProjLod(sampler1DShadow, vec4, float *lod*)
vec4 shadow2DProjLod(sampler2DShadow, vec4, float *lod*)

Noise Functions (8.9 p60)
float noise1(genType)
vec2 noise2(genType)
vec3 noise3(genType)
vec4 noise4(genType)

Type Properties

- Matrix / vector / integer / floating point types.
- *Strict* casting requirements.
- Const / attribute / varying / uniform variables.

Control Loops in GLSL

```
if (condition) {
   statements;
} else {
   statements;
}
```

Beware:
If Statement
Implementation

```
for (int i=0 ; i<10 ; i++) {
   statements;
}
```

```
while (condition) {
   statements;
}
```

Builtins

VERTEX SHADER VARIABLES

Special Output Variables (7.1 p42) *access=RW*
```
vec4  gl_Position;      shader must write
float  gl_PointSize;    enable GL_VERTEX_PROGRAM_POINT_SIZE
vec4  gl_ClipVertex;
```

Attribute Inputs (7.3 p44) *access=RO*
```
attribute vec4  gl_Vertex;
attribute vec3  gl_Normal;
attribute vec4  gl_Color;
attribute vec4  gl_SecondaryColor;
attribute vec4  gl_MultiTexCoord0;
attribute vec4  gl_MultiTexCoord1;
attribute vec4  gl_MultiTexCoord2;
attribute vec4  gl_MultiTexCoord3;
attribute vec4  gl_MultiTexCoord4;
attribute vec4  gl_MultiTexCoord5;
attribute vec4  gl_MultiTexCoord6;
attribute vec4  gl_MultiTexCoord7;
attribute float  gl_FogCoord;
```

Varying Outputs (7.6 p48) *access=RW*
```
varying vec4  gl_FrontColor;
varying vec4  gl_BackColor;  enable GL_VERTEX_PROGRAM_TWO_SIDE
varying vec4  gl_FrontSecondaryColor;
varying vec4  gl_BackSecondaryColor;
varying vec4  gl_TexCoord[ ];      MAX=gl_MaxTextureCoords
varying float  gl_FogFragCoord;
```

FRAGMENT SHADER VARIABLES

Special Output Variables (7.2 p43) *access=RW*
```
vec4  gl_FragColor;
vec4  gl_FragData[gl_MaxDrawBuffers];
float  gl_FragDepth;      DEFAULT=glFragCoord.z
```

Varying Inputs (7.6 p48) *access=RO*
```
varying vec4  gl_Color;
varying vec4  gl_SecondaryColor;
varying vec4  gl_TexCoord[ ];      MAX=gl_MaxTextureCoords
varying float  gl_FogFragCoord;
```

Special Input Variables (7.2 p43) *access=RO*
```
vec4  gl_FragCoord;              pixel coordinates
bool  gl_FrontFacing;
```

BUILT-IN CONSTANTS (7.4 p44)

```
const int gl_MaxVertexUniformComponents;
const int gl_MaxFragmentUniformComponents;
const int gl_MaxVertexAttribs;
const int gl_MaxVaryingFloats;
const int gl_MaxDrawBuffers;
const int gl_MaxTextureCoords;
const int gl_MaxTextureUnits;
const int gl_MaxTextureImageUnits;
const int gl_MaxVertexTextureImageUnits;
const int gl_MaxCombinedTextureImageUnits;
const int gl_MaxLights;
const int gl_MaxClipPlanes;
```

BUILT-IN UNIFORMs (7.5 p45) *access=RO*

```
uniform mat4  gl_ModelViewMatrix;
uniform mat4  gl_ModelViewProjectionMatrix;
uniform mat4  gl_ProjectionMatrix;
uniform mat4  gl_TextureMatrix[gl_MaxTextureCoords];

uniform mat4  gl_ModelViewMatrixInverse;
uniform mat4  gl_ModelViewProjectionMatrixInverse;
uniform mat4  gl_ProjectionMatrixInverse;
uniform mat4  gl_TextureMatrixInverse[gl_MaxTextureCoords];

uniform mat4  gl_ModelViewMatrixTranspose;
uniform mat4  gl_ModelViewProjectionMatrixTranspose;
uniform mat4  gl_ProjectionMatrixTranspose;
uniform mat4  gl_TextureMatrixTranspose[gl_MaxTextureCoords];

uniform mat4  gl_ModelViewMatrixInverseTranspose;
uniform mat4  gl_ModelViewProjectionMatrixInverseTranspose;
uniform mat4  gl_ProjectionMatrixInverseTranspose;
uniform mat4  gl_TextureMatrixInverseTranspose[gl_MaxTextureCoords];

uniform mat3  gl_NormalMatrix;
uniform float   gl_NormalScale;

struct gl_DepthRangeParameters {
   float near;
   float far;
   float diff;
};
uniform gl_DepthRangeParameters gl_DepthRange;

struct  gl_FogParameters {
   vec4  color;
   float  density;
   float  start;
   float  end;
   float  scale;
};
uniform gl_FogParameters gl_Fog;

struct gl_LightSourceParameters {
   vec4  ambient;
   vec4  diffuse;
   vec4  specular;
   vec4  position;
   vec4  halfVector;
   vec3  spotDirection;
   float  spotExponent;
   float  spotCutoff;
   float  spotCosCutoff;
   float  constantAttenuation;
   float  linearAttenuation;
   float  quadraticAttenuation;
};
uniform gl_LightSourceParameters gl_LightSource[gl_MaxLights];

struct  gl_LightModelParameters {
   vec4  ambient;
};
uniform gl_LightModelParameters gl_LightModel;

struct  gl_LightModelProducts {
   vec4  sceneColor;
};
uniform gl_LightModelProducts  gl_FrontLightModelProduct;
uniform gl_LightModelProducts  gl_BackLightModelProduct;

struct  gl_LightProducts {
   vec4  ambient;
   vec4  diffuse;
   vec4  specular;
};
uniform gl_LightProducts gl_FrontLightProduct[gl_MaxLights];
uniform gl_LightProducts gl_BackLightProduct[gl_MaxLights];

struct gl_MaterialParameters {
   vec4  emission;
   vec4  ambient;
   vec4  diffuse;
   vec4  specular;
   float  shininess;
};
uniform gl_MaterialParameters gl_FrontMaterial;
uniform gl_MaterialParameters gl_BackMaterial;

struct gl_PointParameters {
   float  size;
   float  sizeMin;
   float  sizeMax;
   float  fadeThresholdSize;
   float  distanceConstantAttenuation;
   float  distanceLinearAttenuation;
   float  distanceQuadraticAttenuation;
};
uniform gl_PointParameters gl_Point;

uniform vec4  gl_TextureEnvColor[gl_MaxTextureUnits];  (1)

uniform vec4  gl_ClipPlane[gl_MaxClipPlanes];

uniform vec4  gl_EyePlaneS[gl_MaxTextureCoords];
uniform vec4  gl_EyePlaneT[gl_MaxTextureCoords];
uniform vec4  gl_EyePlaneR[gl_MaxTextureCoords];
uniform vec4  gl_EyePlaneQ[gl_MaxTextureCoords];

uniform vec4  gl_ObjectPlaneS[gl_MaxTextureCoords];
uniform vec4  gl_ObjectPlaneT[gl_MaxTextureCoords];
uniform vec4  gl_ObjectPlaneR[gl_MaxTextureCoords];
uniform vec4  gl_ObjectPlaneQ[gl_MaxTextureCoords];
```

OpenSceneGraph Preset Uniforms

as of OSG 1.0
```
int     osg_FrameNumber;
float   osg_FrameTime;
float   osg_DeltaFrameTime;
mat4    osg_ViewMatrix;
mat4    osg_ViewMatrixInverse;
```

Fine print / disclaimer
Copyright 2005 Mike Weiblen http://mew.cx/
Please send feedback/corrections/comments to glsl@mew.cx
OpenGL is a registered trademark of Silicon Graphics Inc.
Except as noted below, if discrepancies between this guide and the GLSL specification, believe the spec!
Revised 2005-11-26

Notes
1. Corrects a typo in the OpenGL 2.0 specification.

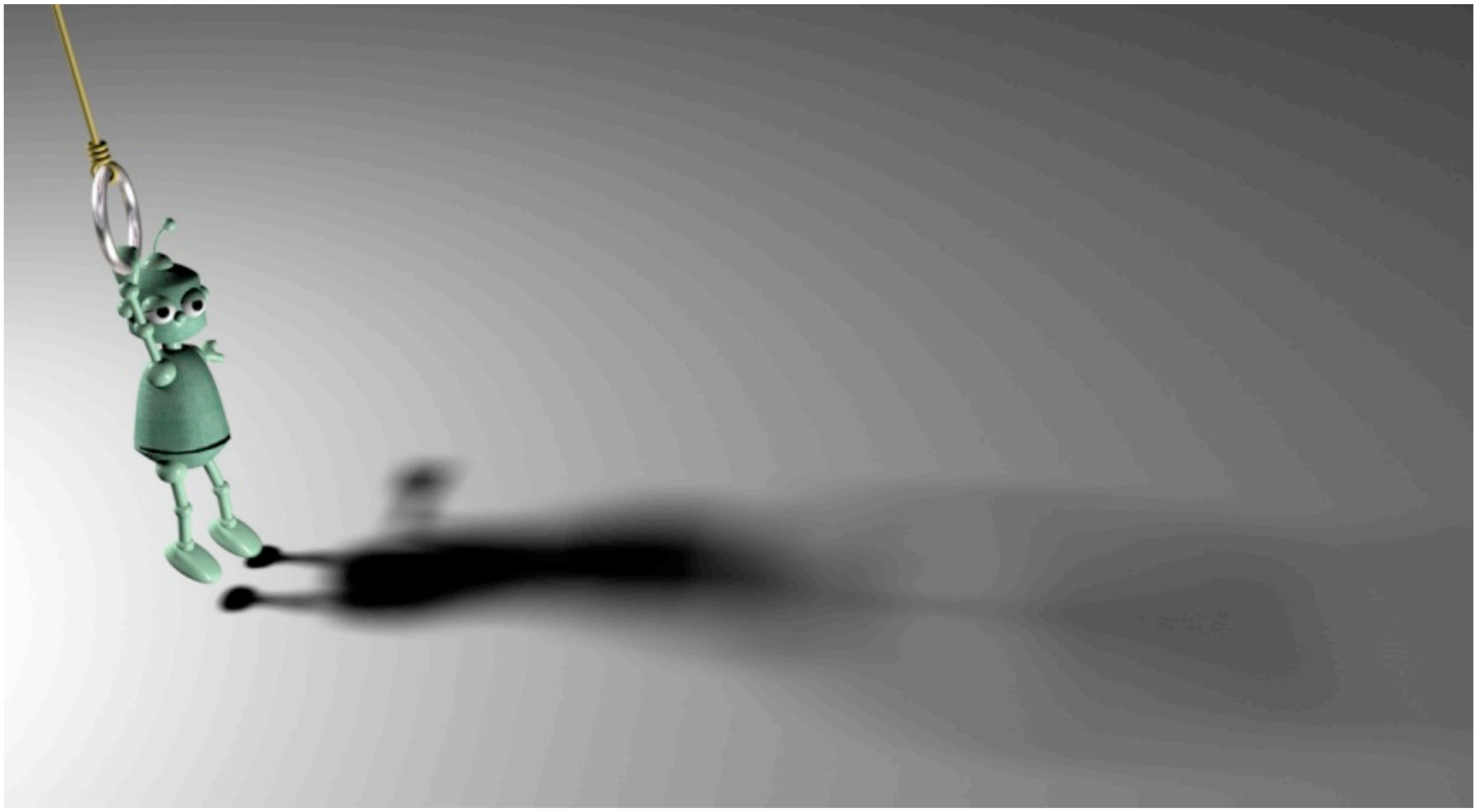
GPU Shading

- Announcements
- GPU Shading Overview
- Example 1: Shadow Maps
- Example 2: Algebraic Surfaces

GPU Shading

- Announcements
- GPU Shading Overview
- Example 1: Shadow Maps
- Example 2: Algebraic Surfaces

Soft Shadows



- Why? How?

Hard Shadows

Shadows

Shadows occur where objects are hidden from a light source

omit any intensity contribution from hidden light sources

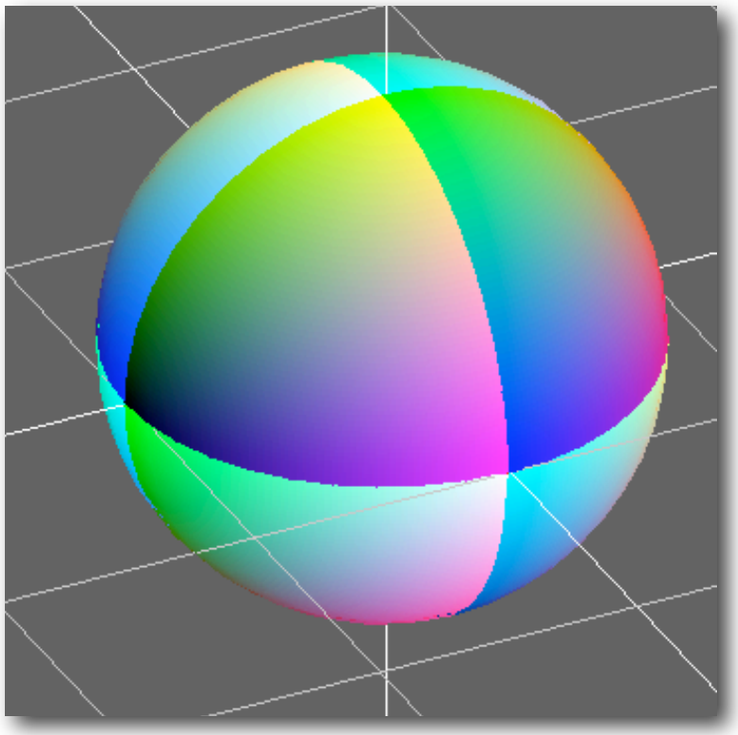
How does the z-buffer work?

How can we use the z-buffer to compute shadows?

GPU Shading

- Announcements
- GPU Shading Overview
- Example 1: Shadow Maps
- Example 2: Algebraic Surfaces

GPU Shading

- Announcements
- GPU Shading Overview
- Example 1: Shadow Maps
- Example 2: Algebraic Surfaces

Happy GPU Coding!