GL2 Shading Language

Bill Licea-Kane
ATI Research, Inc.
bill@ati.com

GL2 Shading Language
Back to the future

"As far as generating pictures from data is concerned, we feel the display processor should be a specialized device, capable only of generating pictures from read-only representations in core."

Myer and Sutherland, On the Design of Display Processors.
***Communications of the ACM, Volume 11 Number 6*, June, 1968**

GL2 Shading Language Back to the future

- June 30, 1992
  - OpenGL 1.0 Final Draft published

GL2 Shading Language
Back to the future

"OpenGL does not provide a programming language. Its function may be controlled by turning operations on or off or specifying parameters to operations, but the rendering algorithms are essentially fixed. One reason for this decision is that, for performance reasons, graphics hardware is usually designed to apply certain operations in a specific order; replacing these operations with arbitrary algorithms is usually infeasible. Programmability would conflict with keeping the API close to the hardware and thus with the goal of maximum performance."

Segal and Akeley, *The Design of the OpenGL Graphics Interface*, 1994

GL2 Shading Language Back to the future

- **August 23, 1993**
  - **First OpenGL 1.0 implementation ships**

GL2 Shading Language
Back to the future

```
billa@entropy.sps.mot.com (William C. Archibald) writes:

|> 1)Quite a few modern rendering techniques and renderers
|>   support shader function calls at arbitrary points
|>   across a surface being rendered.  ...  What I would
|>   _like_ to do is call the shader at visible
|>   (front-facing) points.  Is there _any_ way to do
|>   this kind of thing in OpenGL?

Not with the current API.  One could extend OpenGL in this
manner...
```

Akeley, comp.graphics.opengl posting, 13 Oct 1993

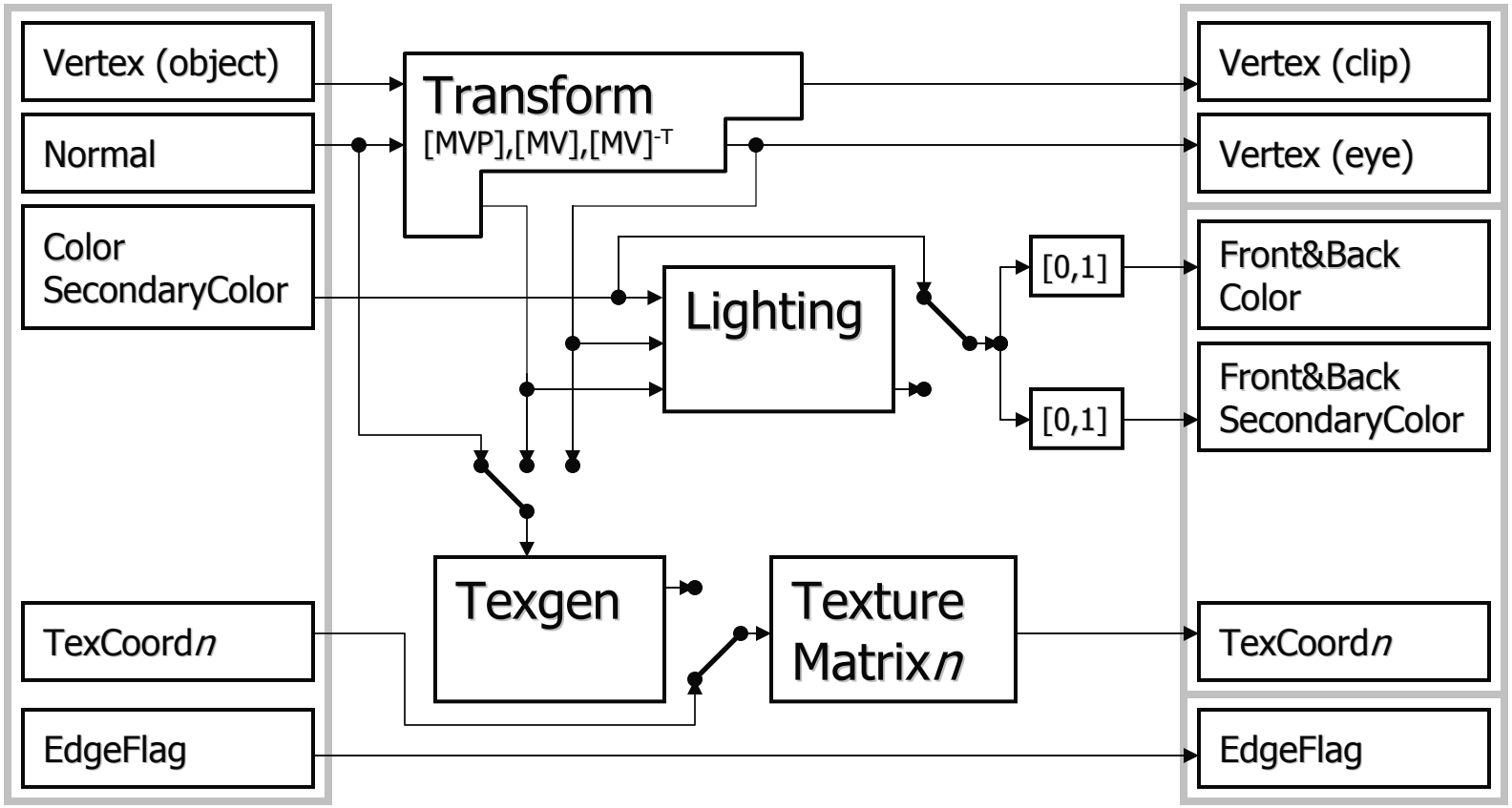
GL2 Shading Language Back to the present

- February 27, 2003
  - arb-gl2 workgroup publishes Draft OpenGL Shading Language
  - End construction zone

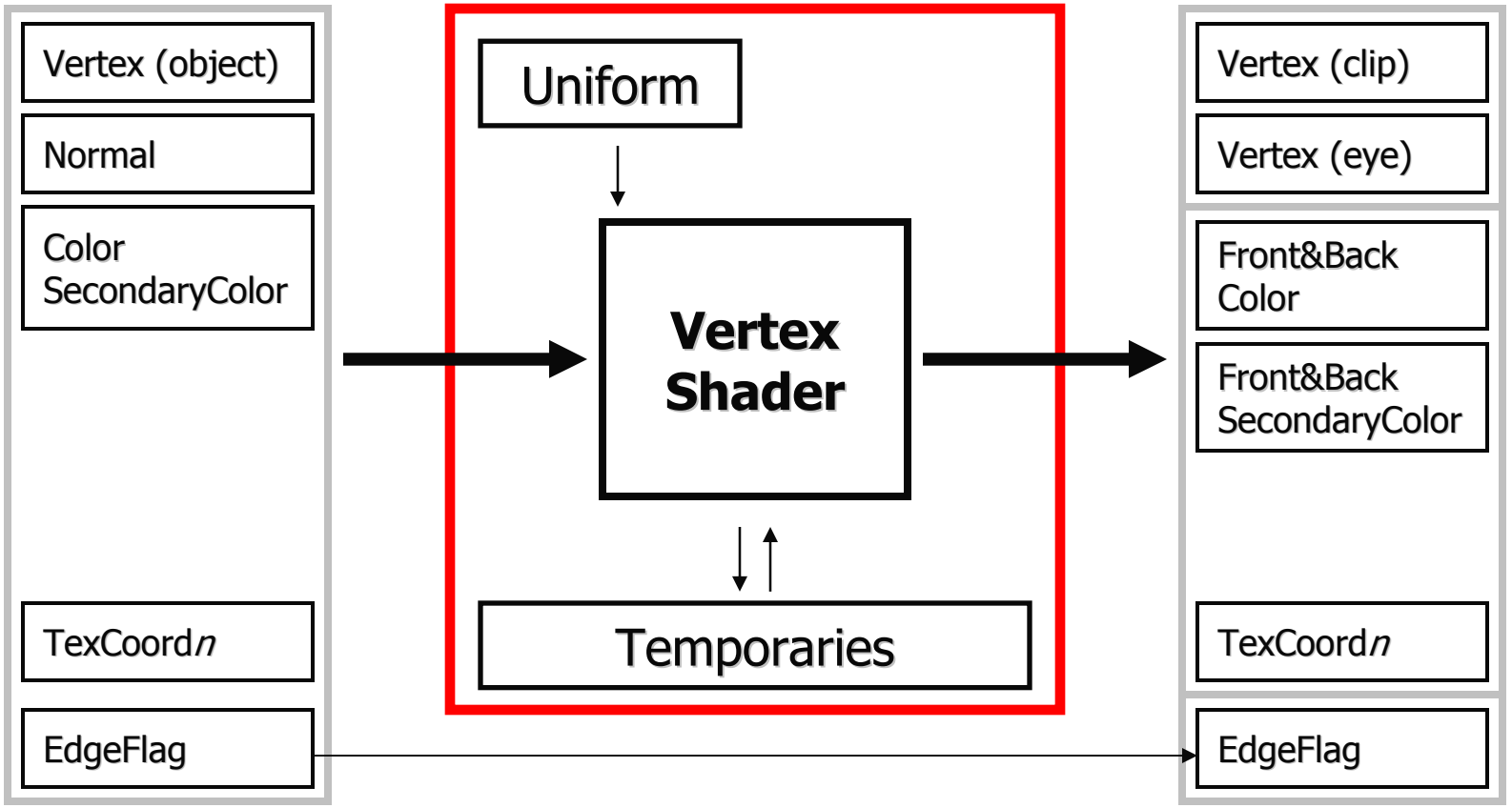
GL2 Shading Language Today

- How we replace Fixed Function
- Review Shading Language
- Shading Language examples
- Procedural Shader demo

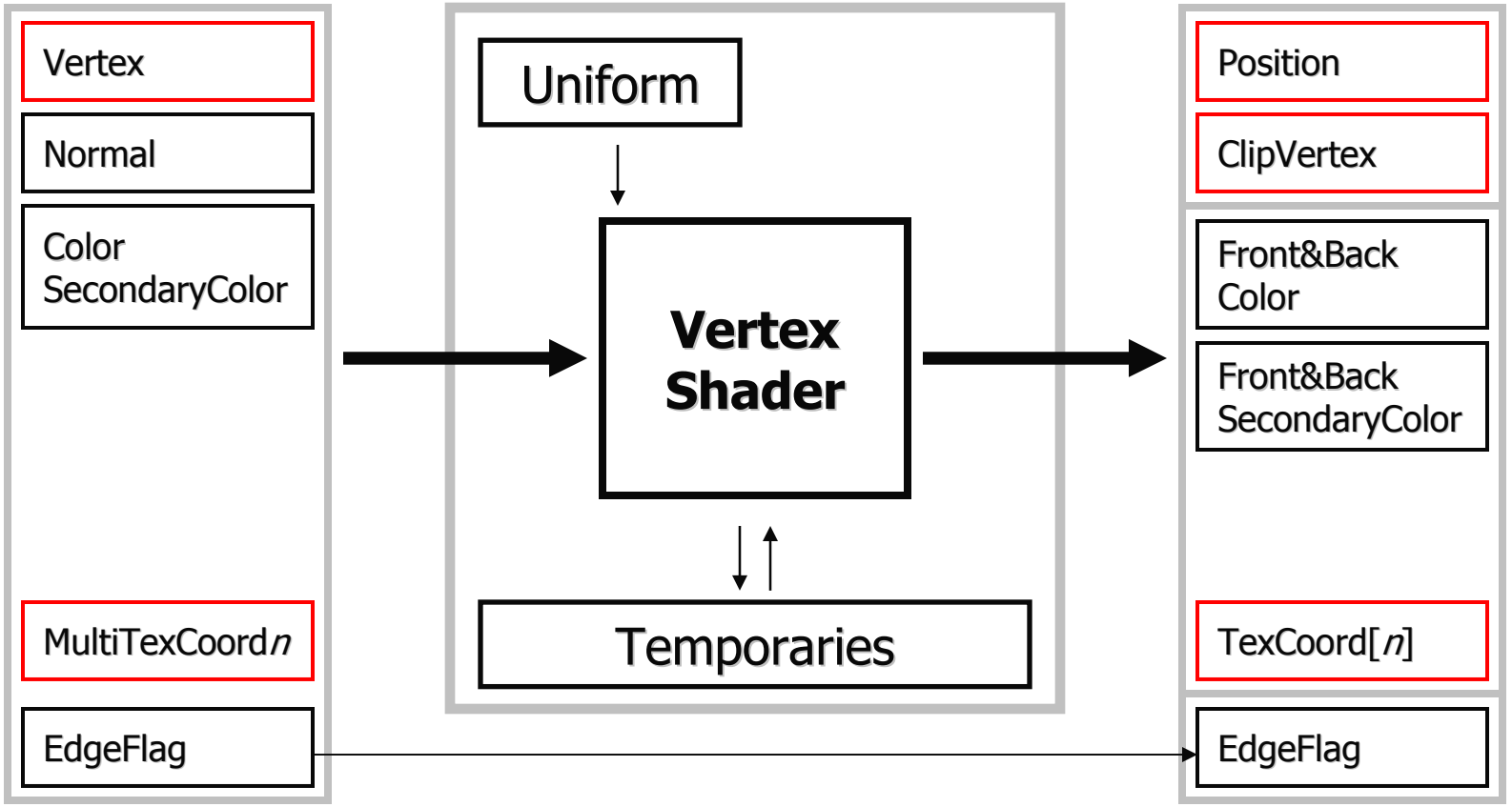
OpenGL
Fixed Function Vertex



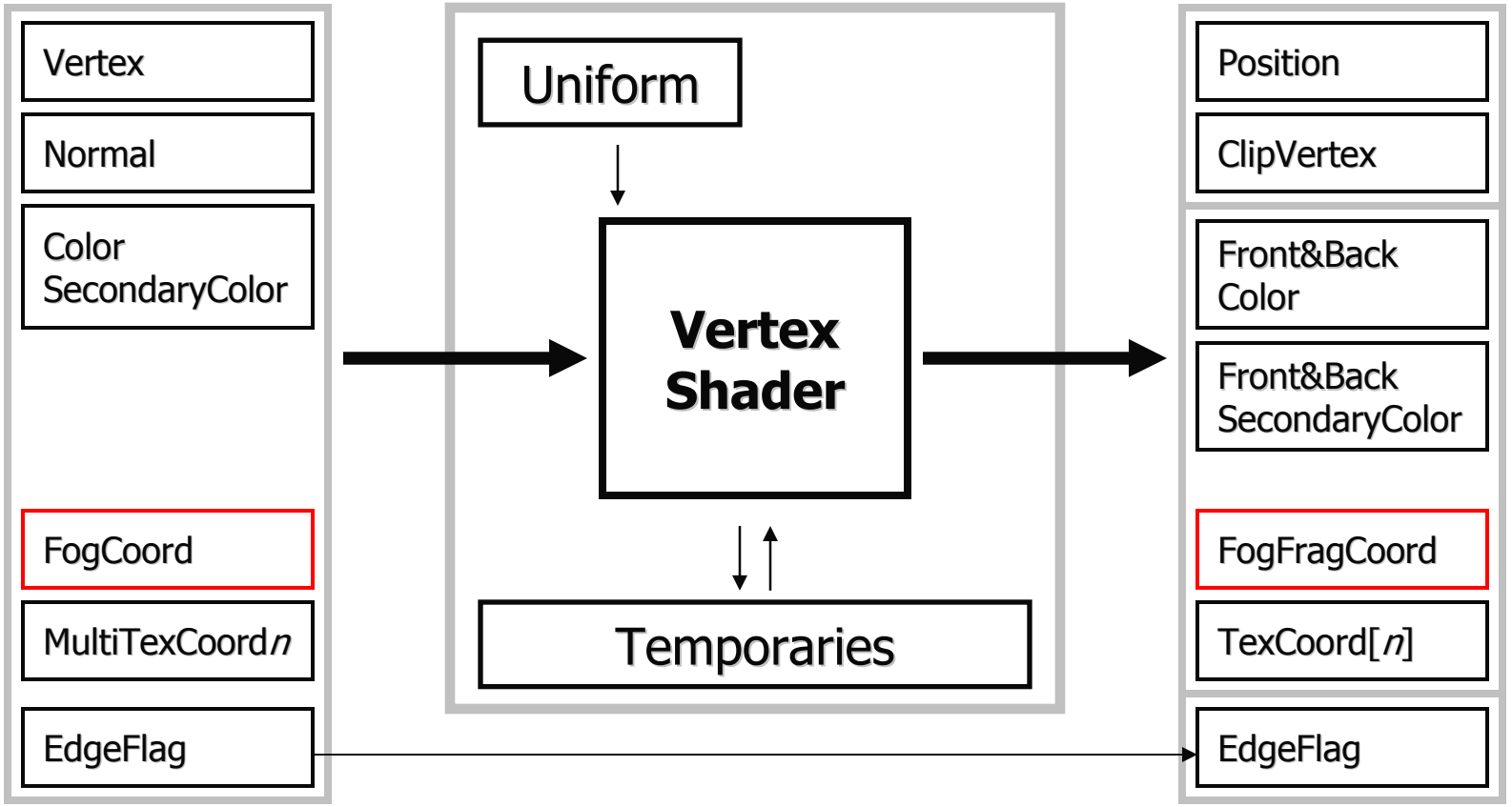
GL2 Vertex Processor



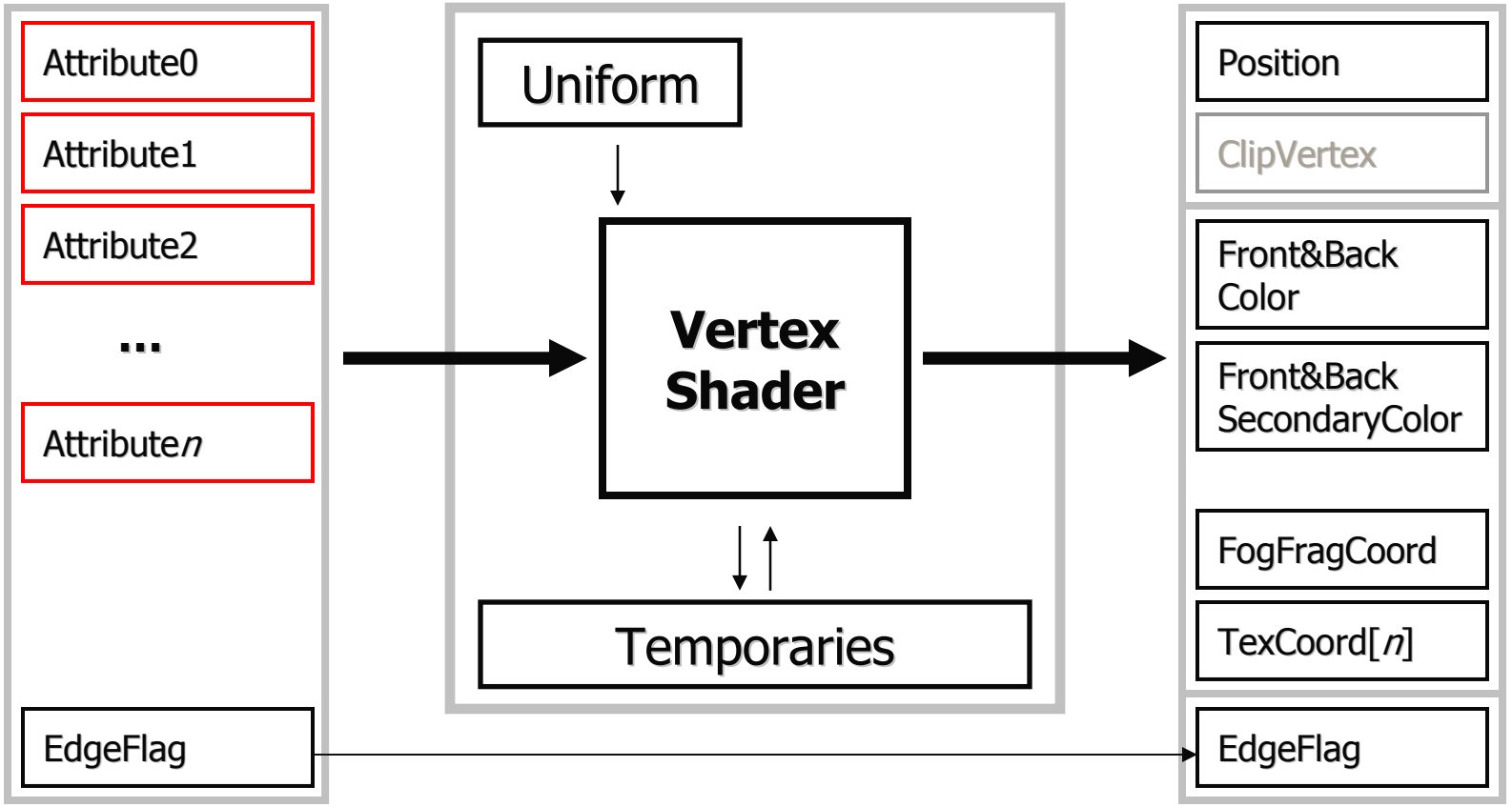
GL2 Vertex Processor Names



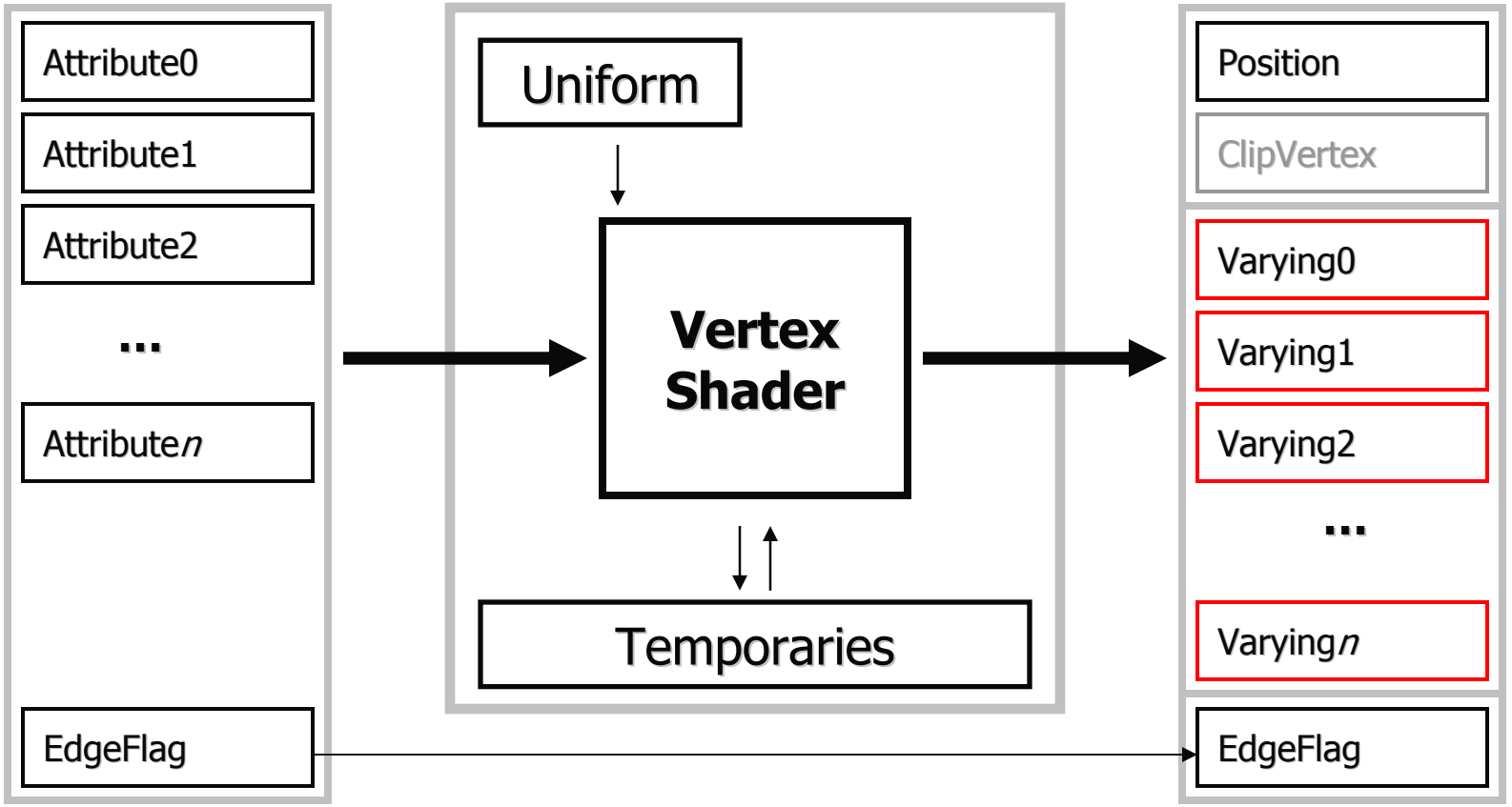
GL2 Vertex Processor
Add Fog Coordinate



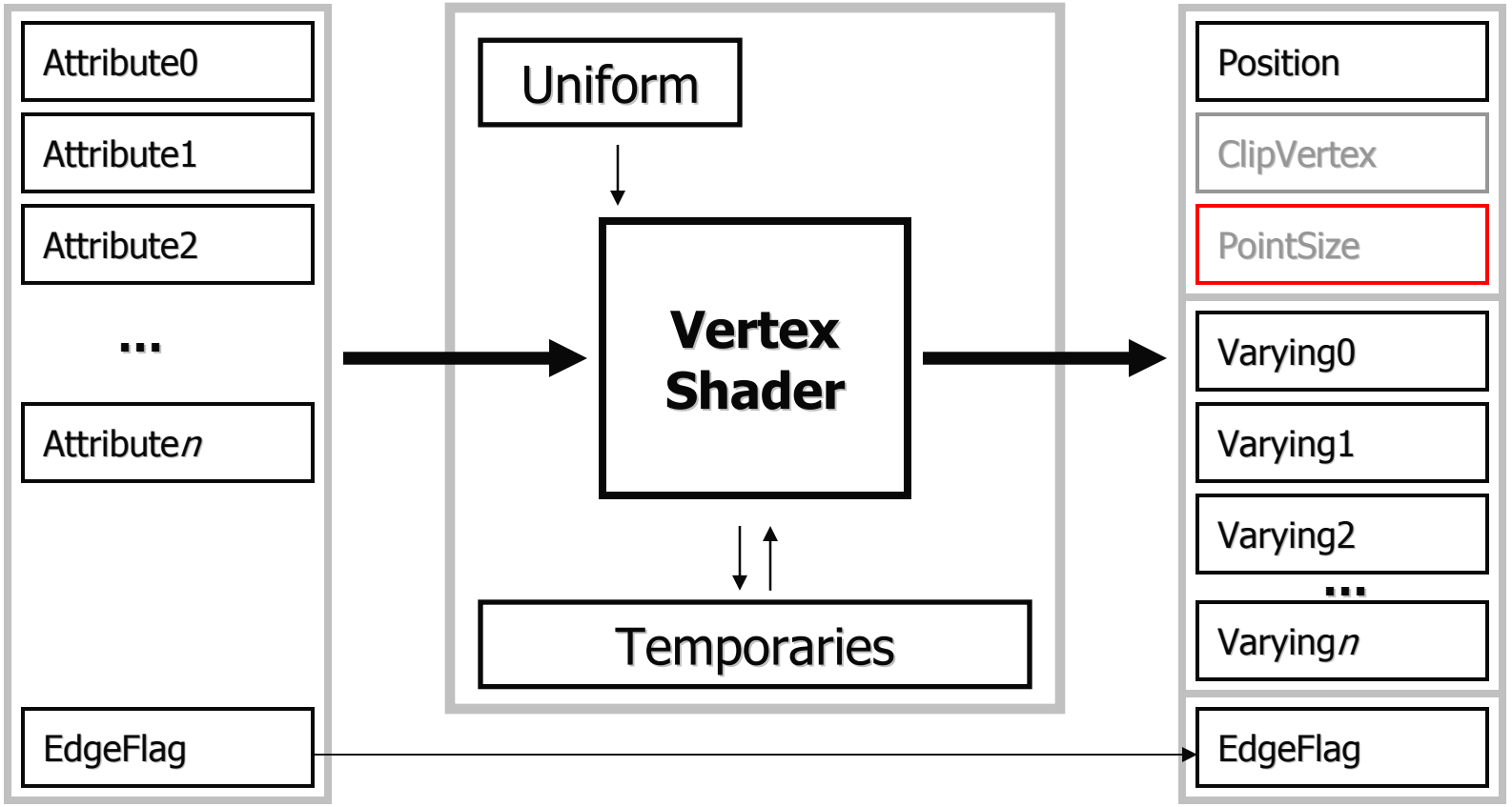
GL2 Vertex Processor Generalize Input



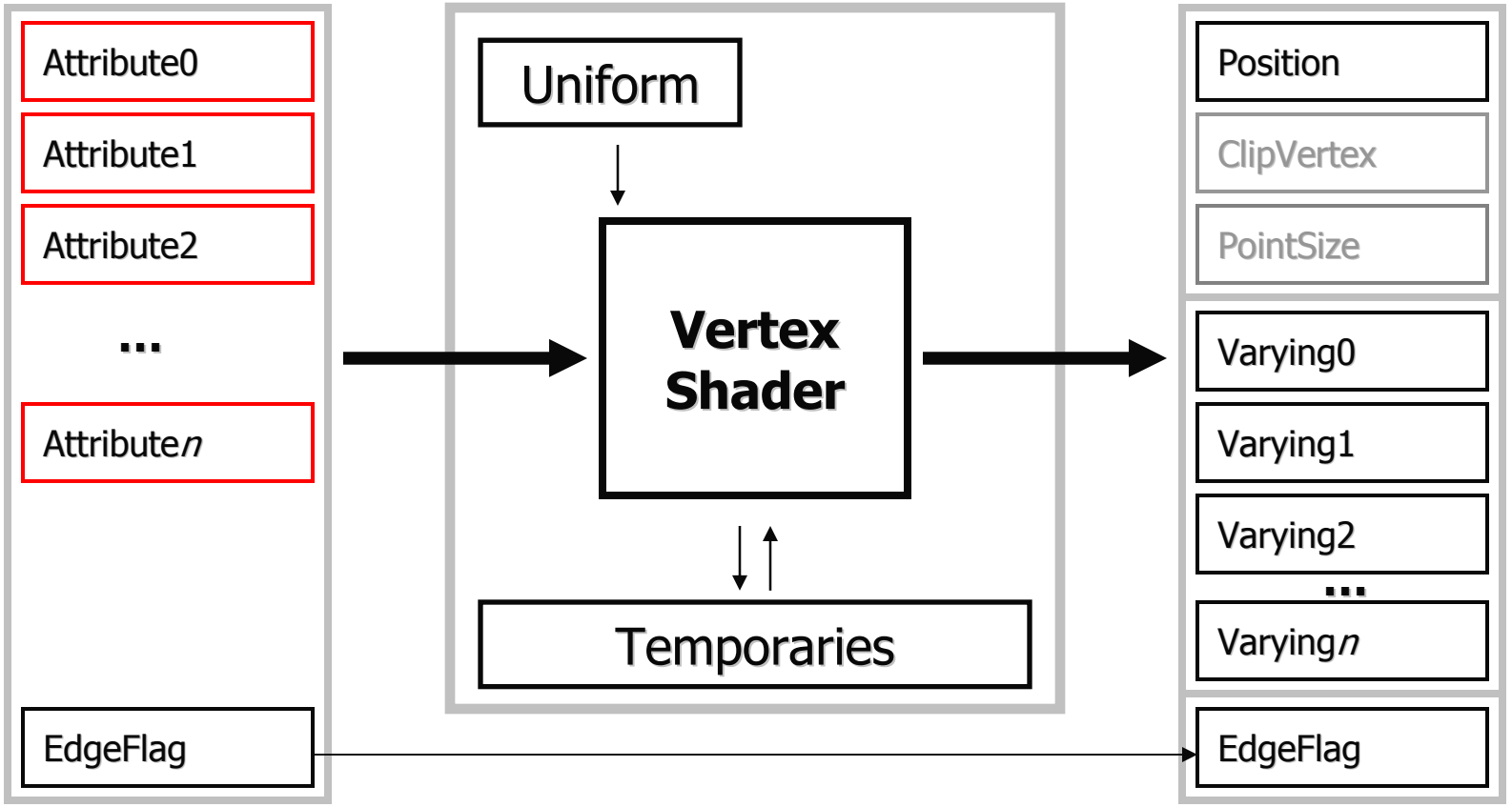
GL2 Vertex Processor Generalize Output



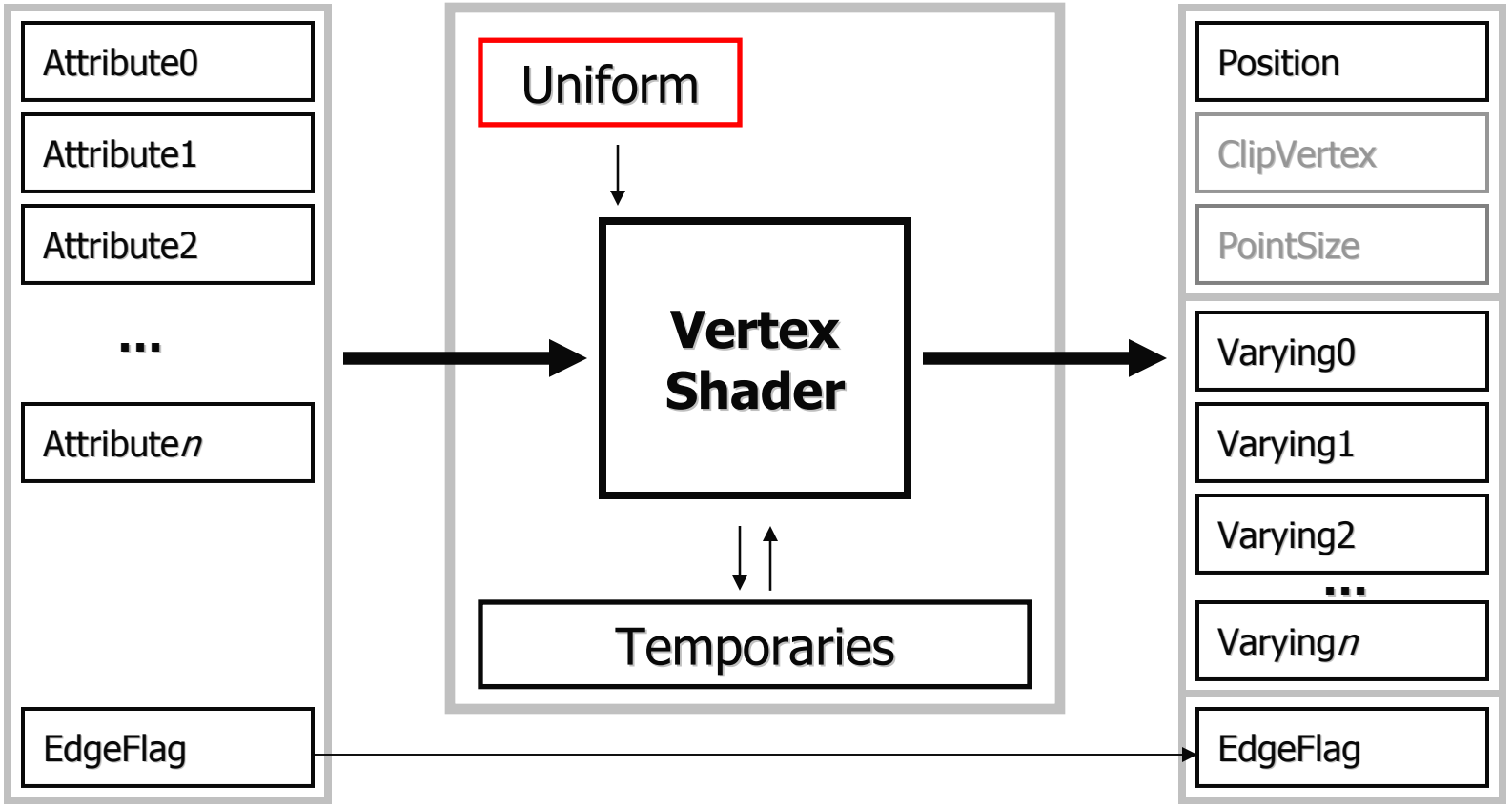
GL2 Vertex Processor Additional Output



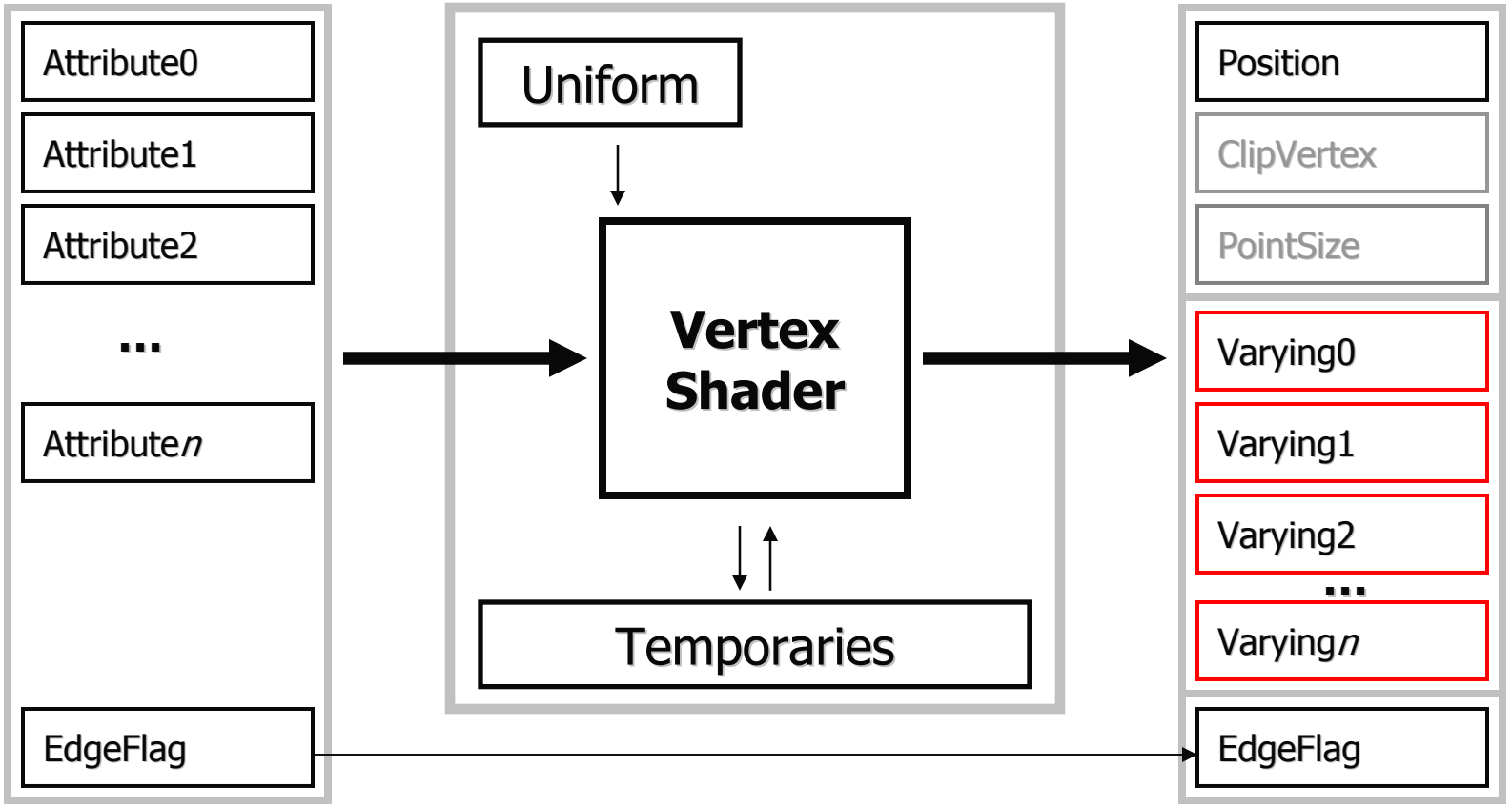
GL2 Vertex Limits
Active Attributes (16 vec4)



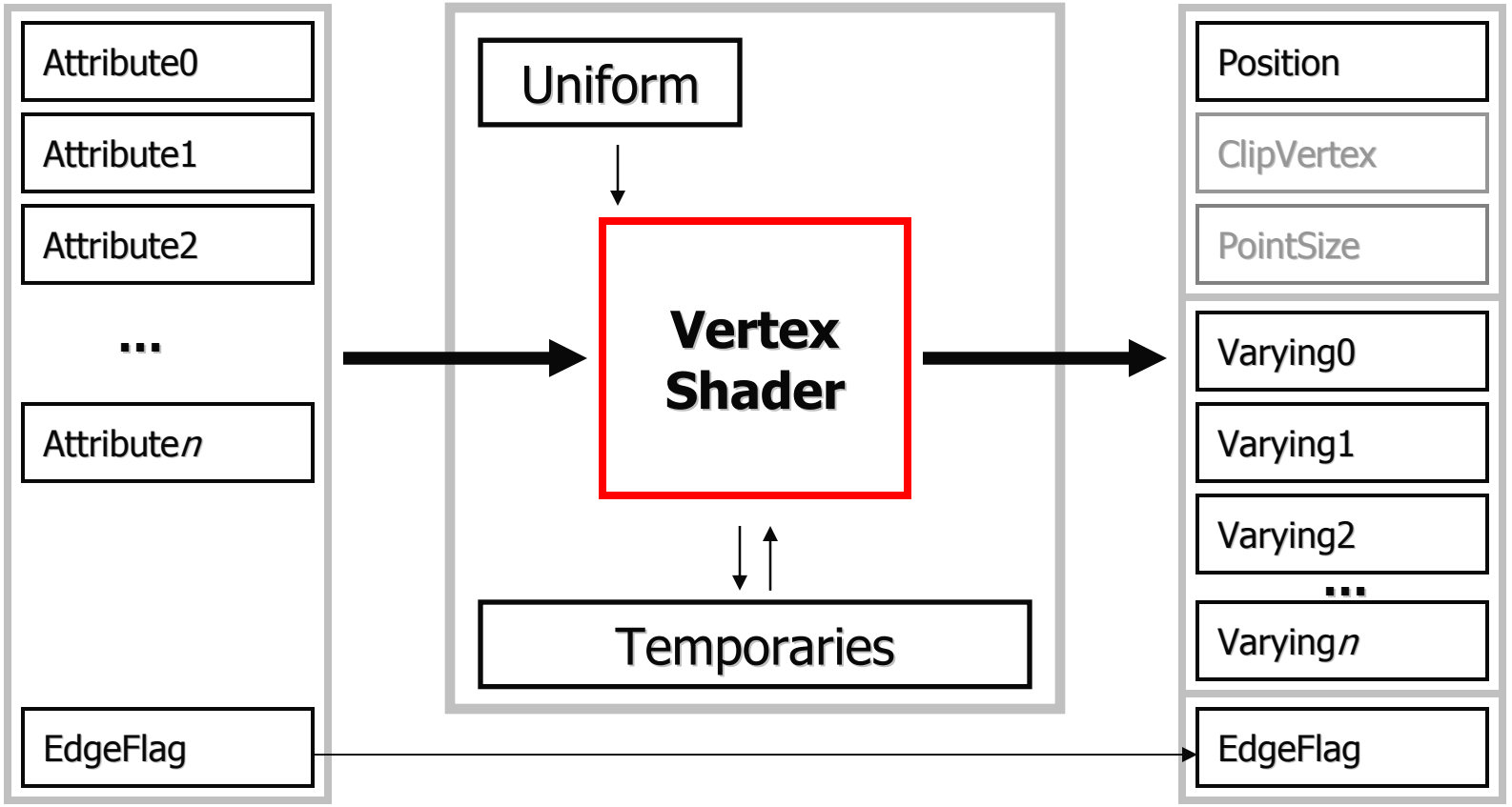
GL2 Vertex Limits
Uniform (512 words)



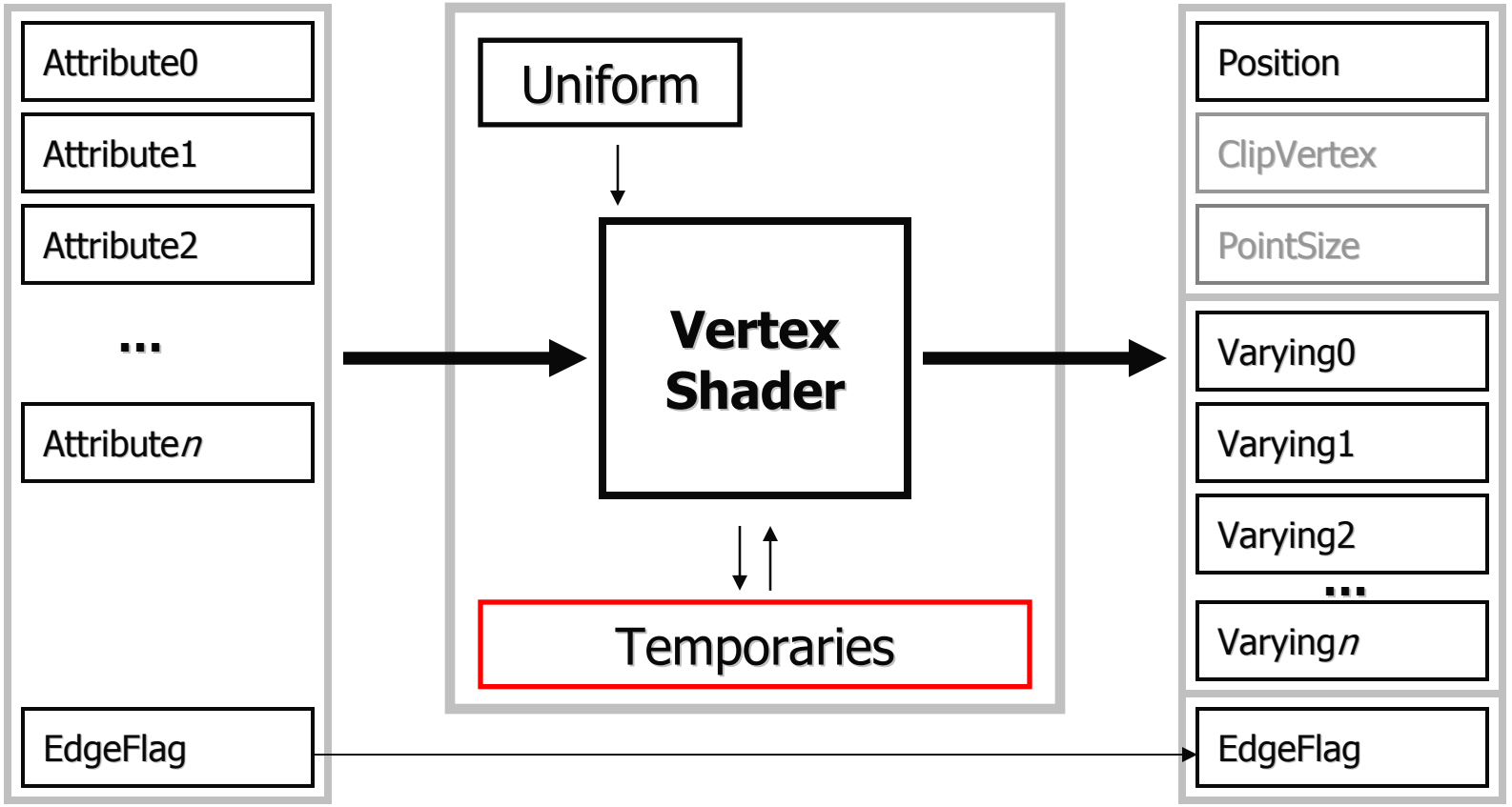
GL2 Vertex Limits
Varying (32 floats)



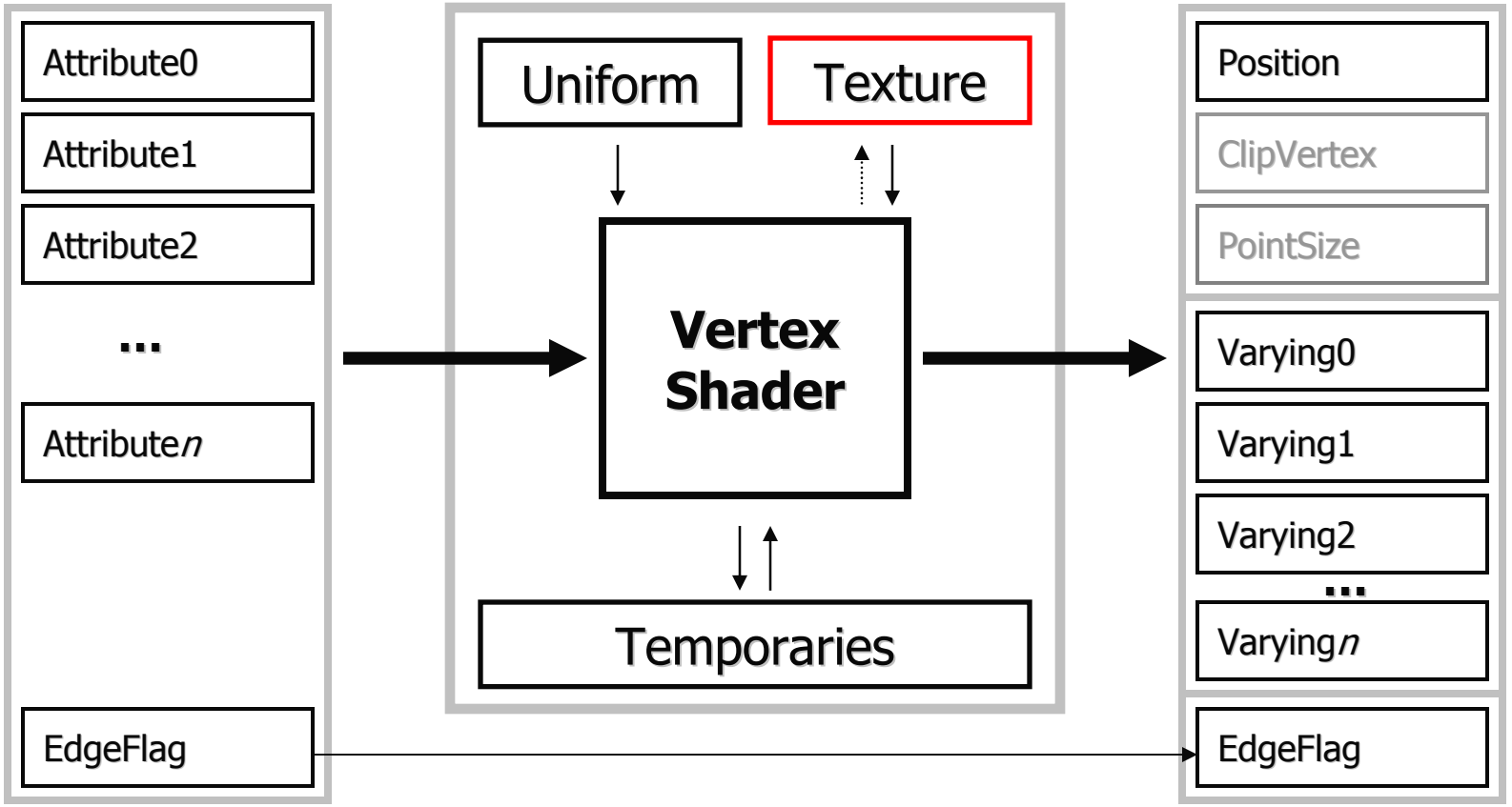
GL2 Vertex Limits Instructions (Virtual)



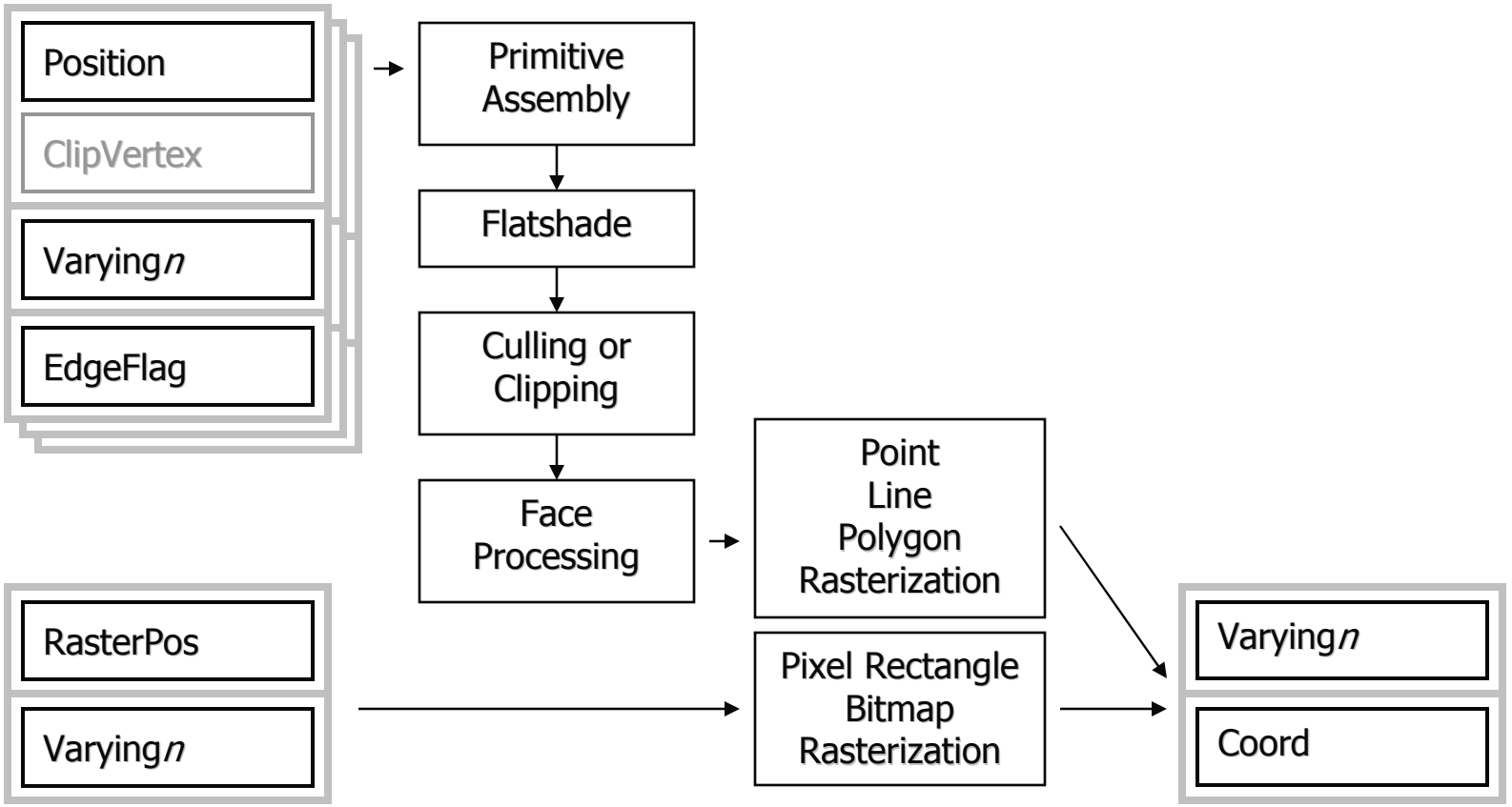
GL2 Vertex Limits Temporaries (Virtual)



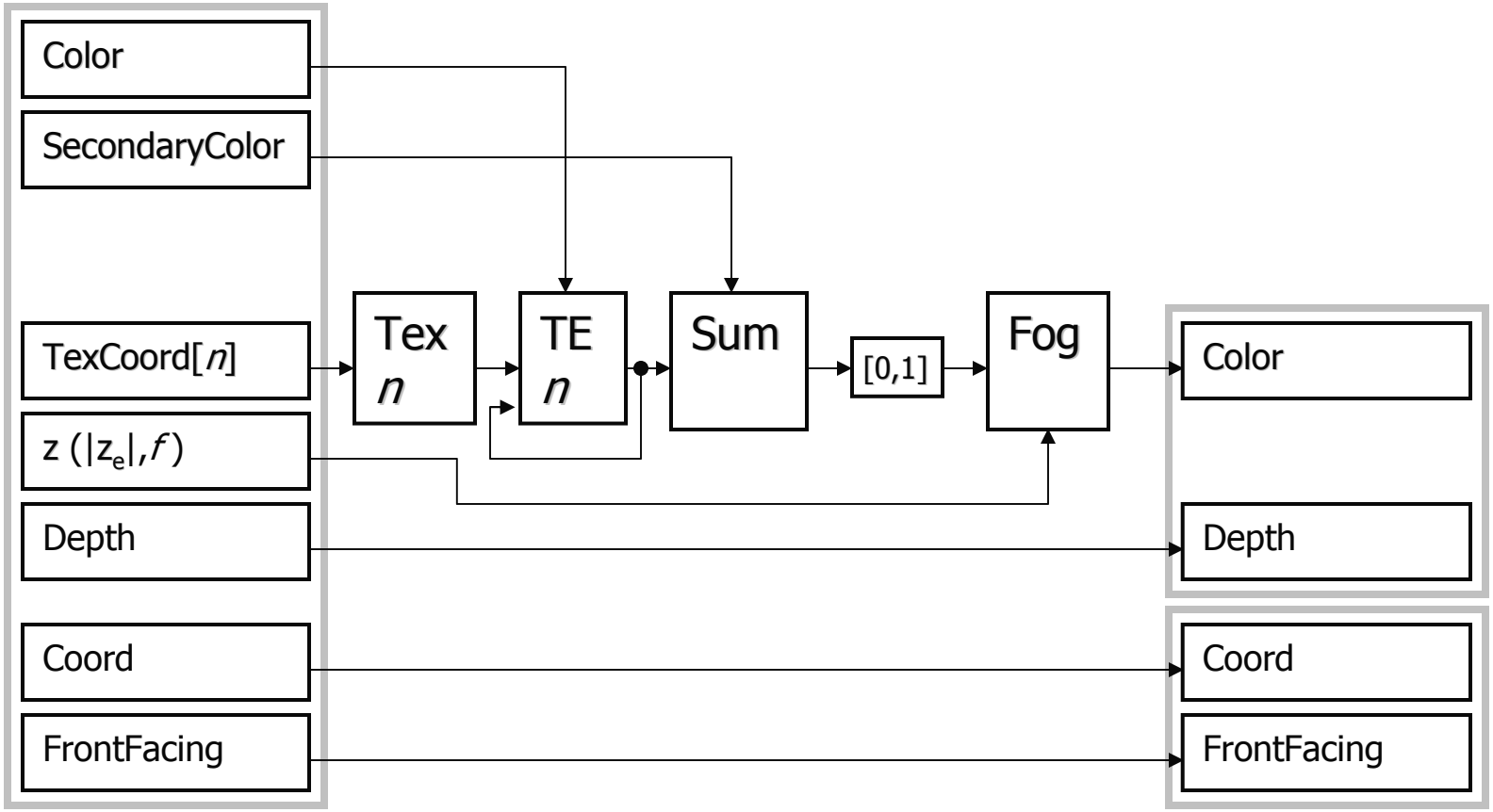
GL2 Vertex Limits
Texture Units (1)



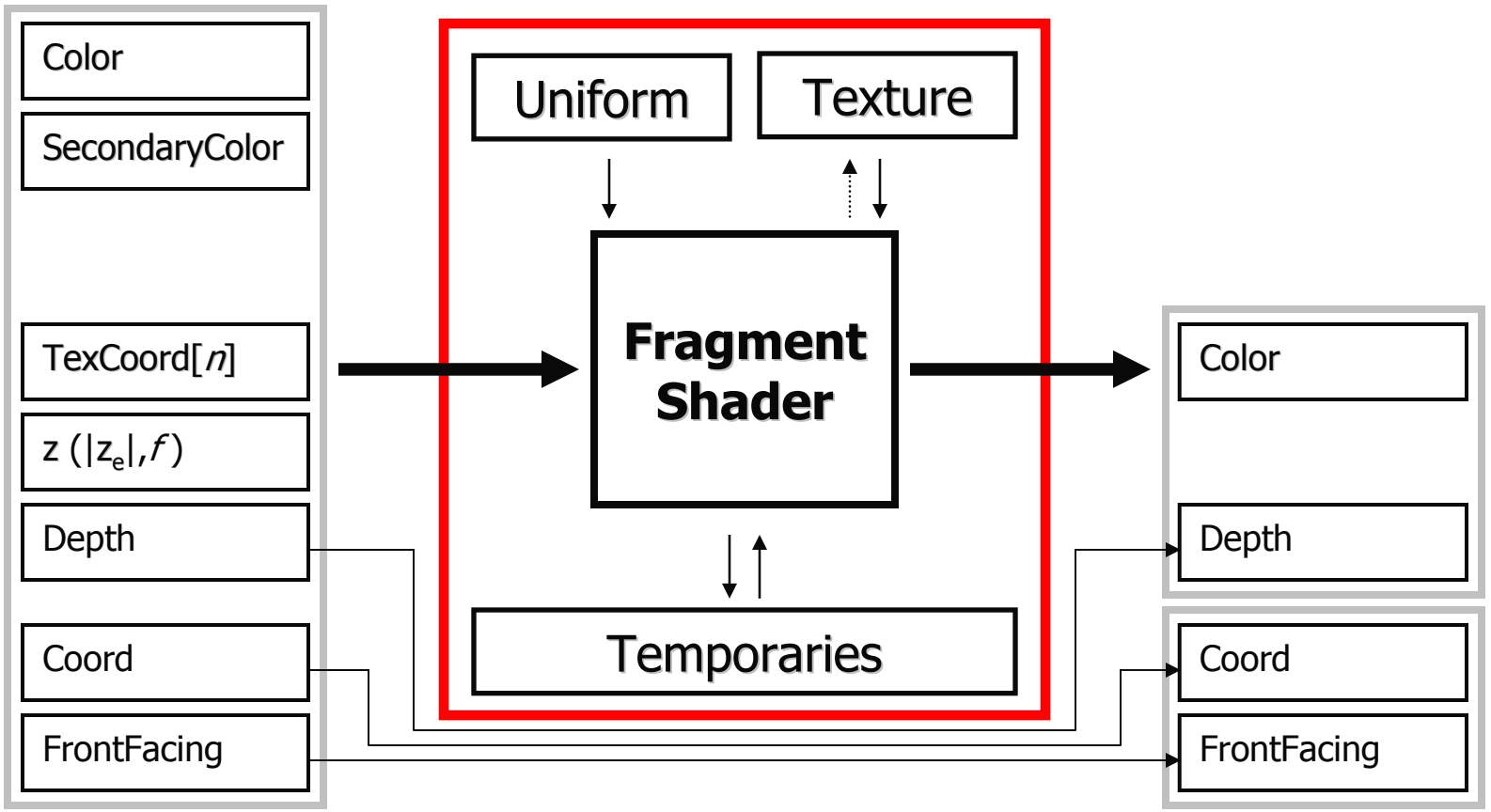
OpenGL
Fixed Function Rasterization



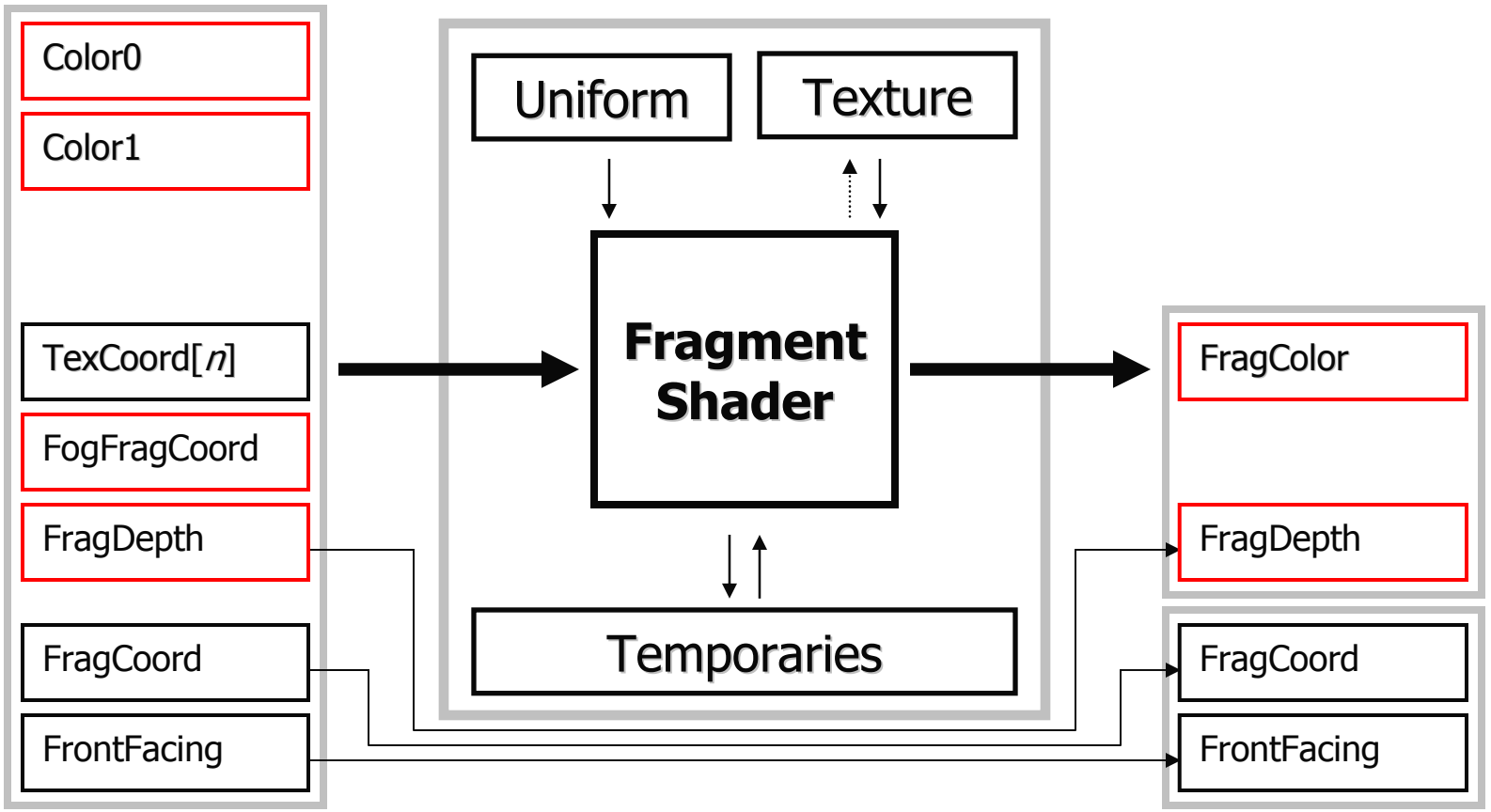
OpenGL
Fixed Function Fragment



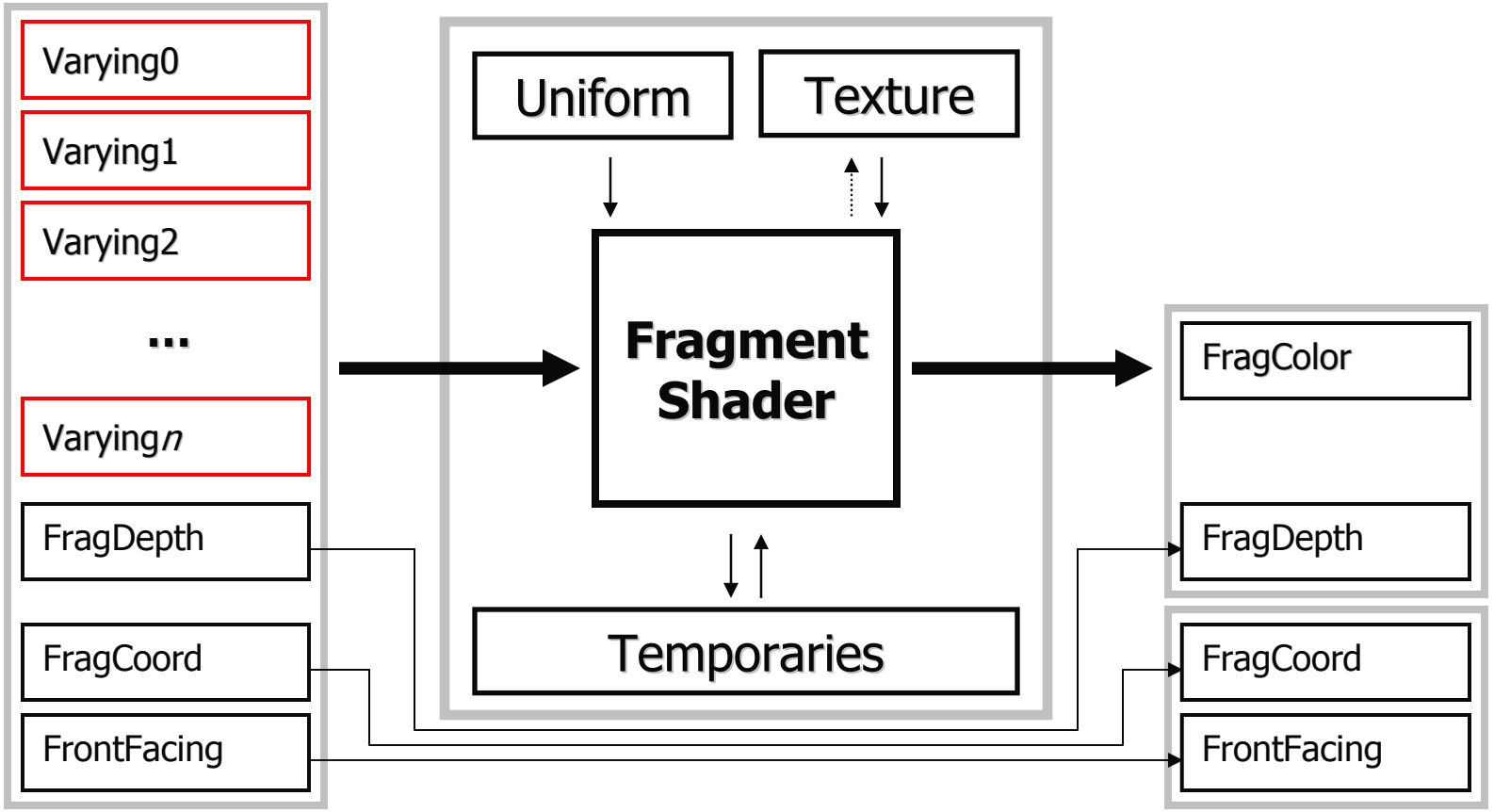
GL2 Fragment Processor



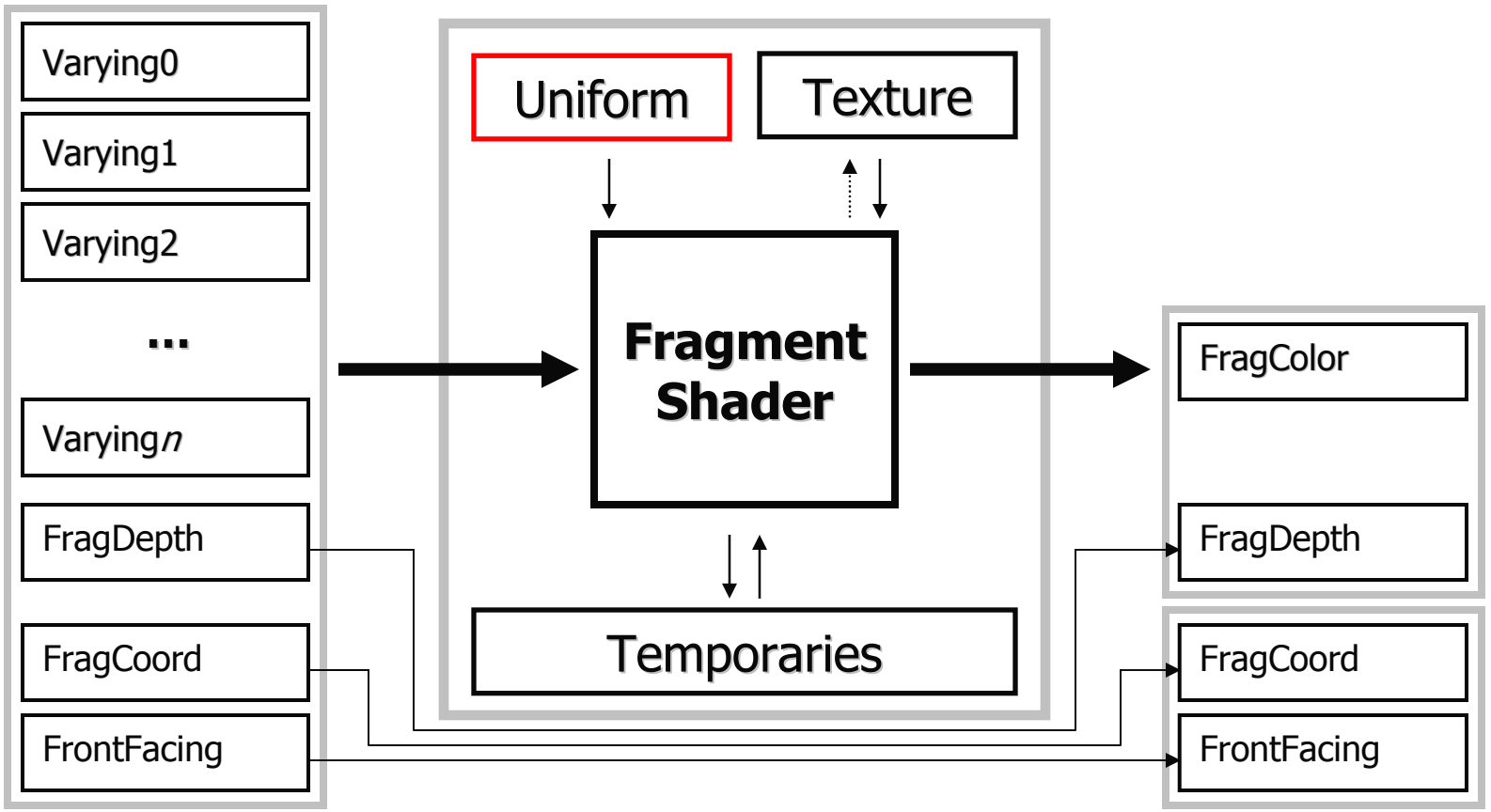
GL2 Fragment Processor Names



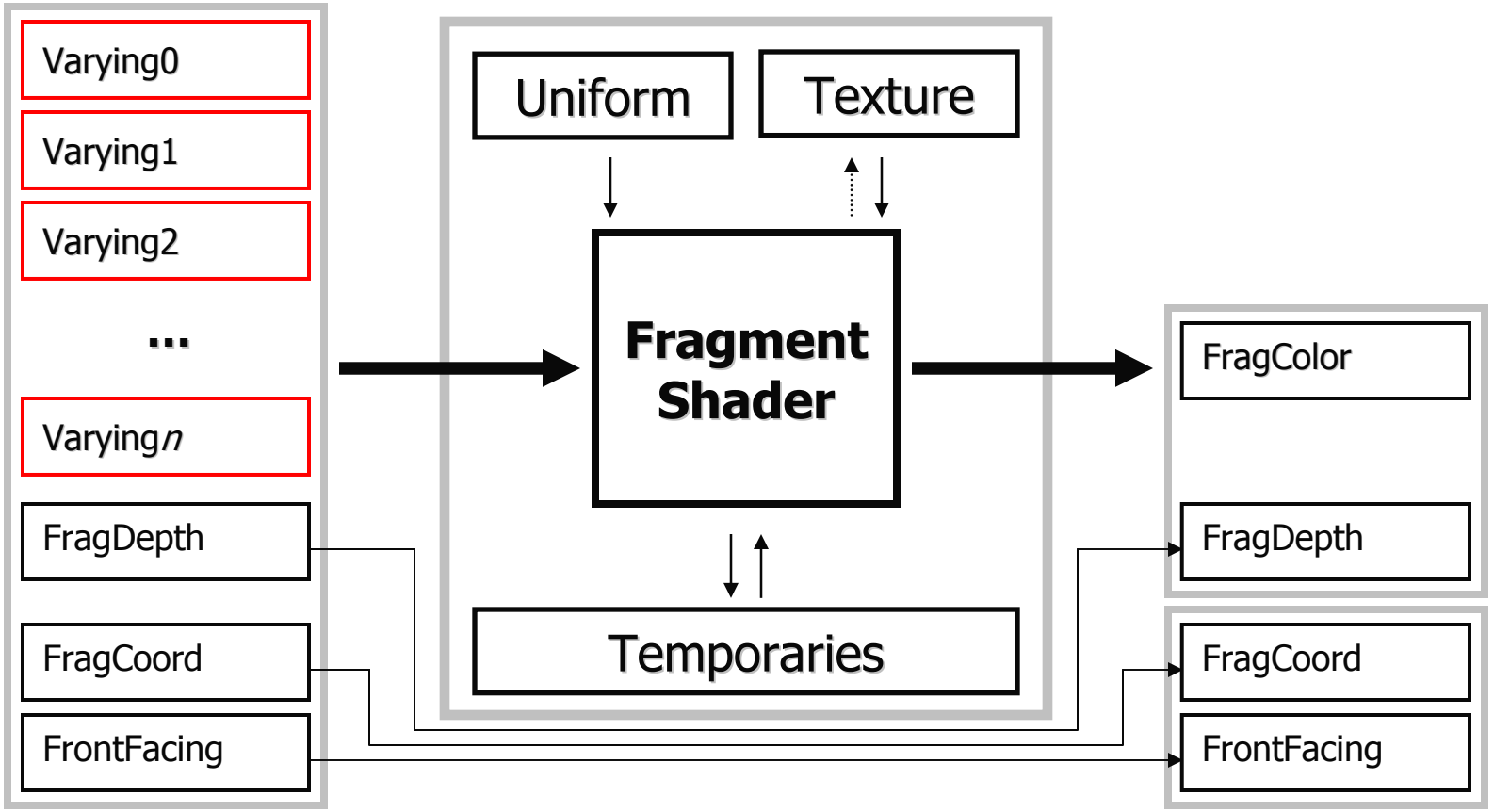
GL2 Fragment Processor Generalize Input



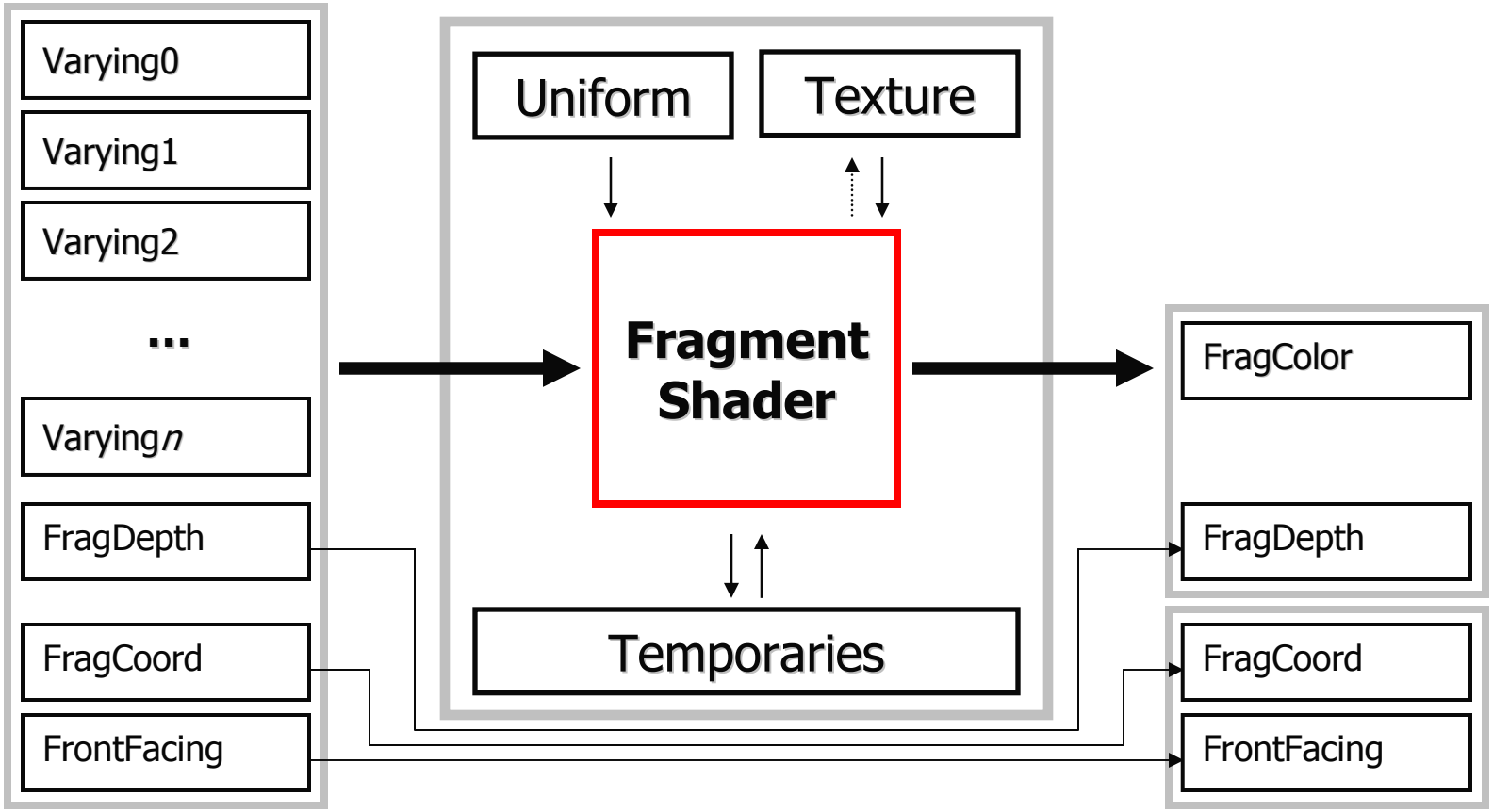
GL2 Fragment Limits Uniforms (64 words)



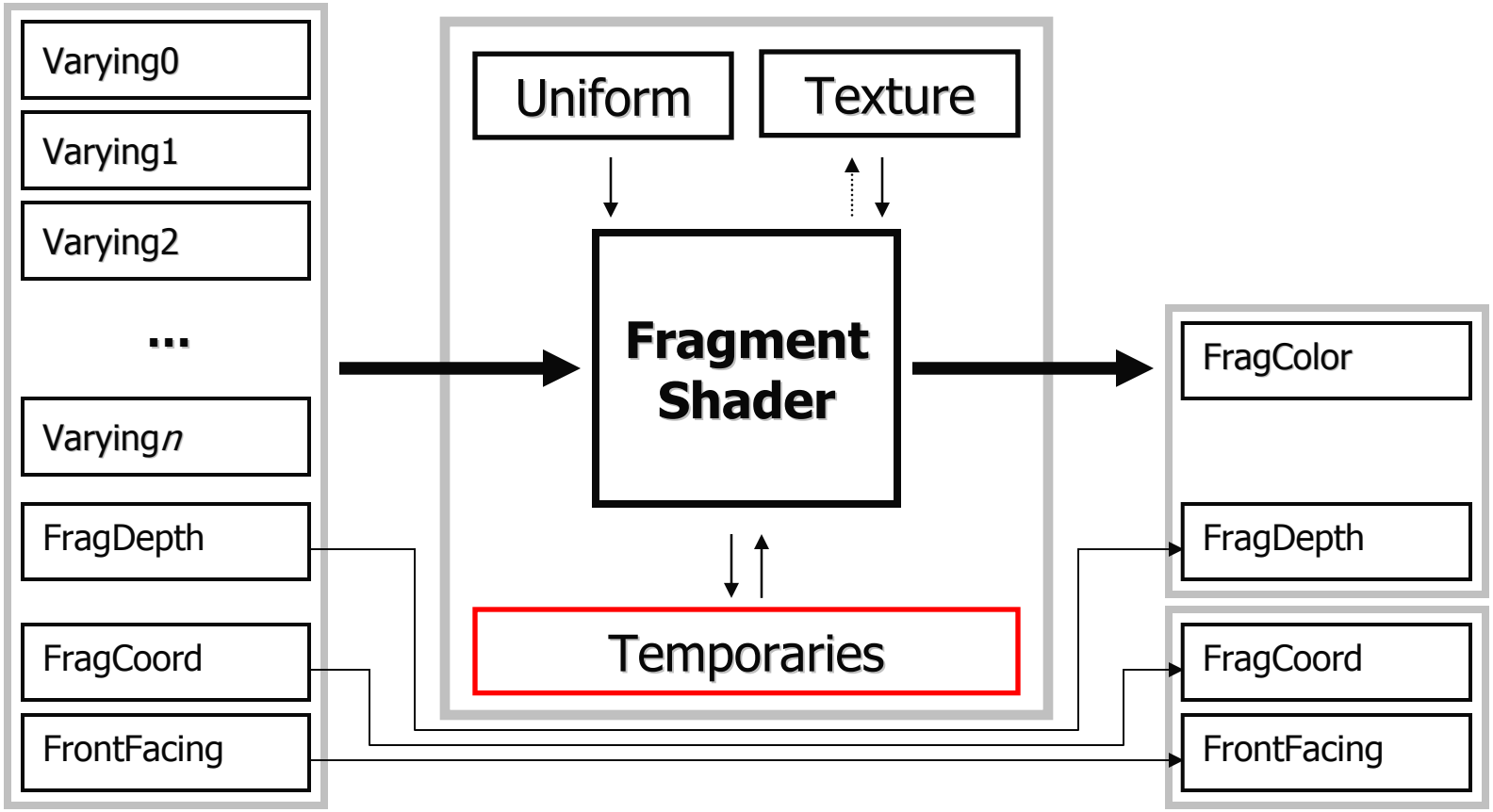
GL2 Fragment Limits
Varying (32 floats)



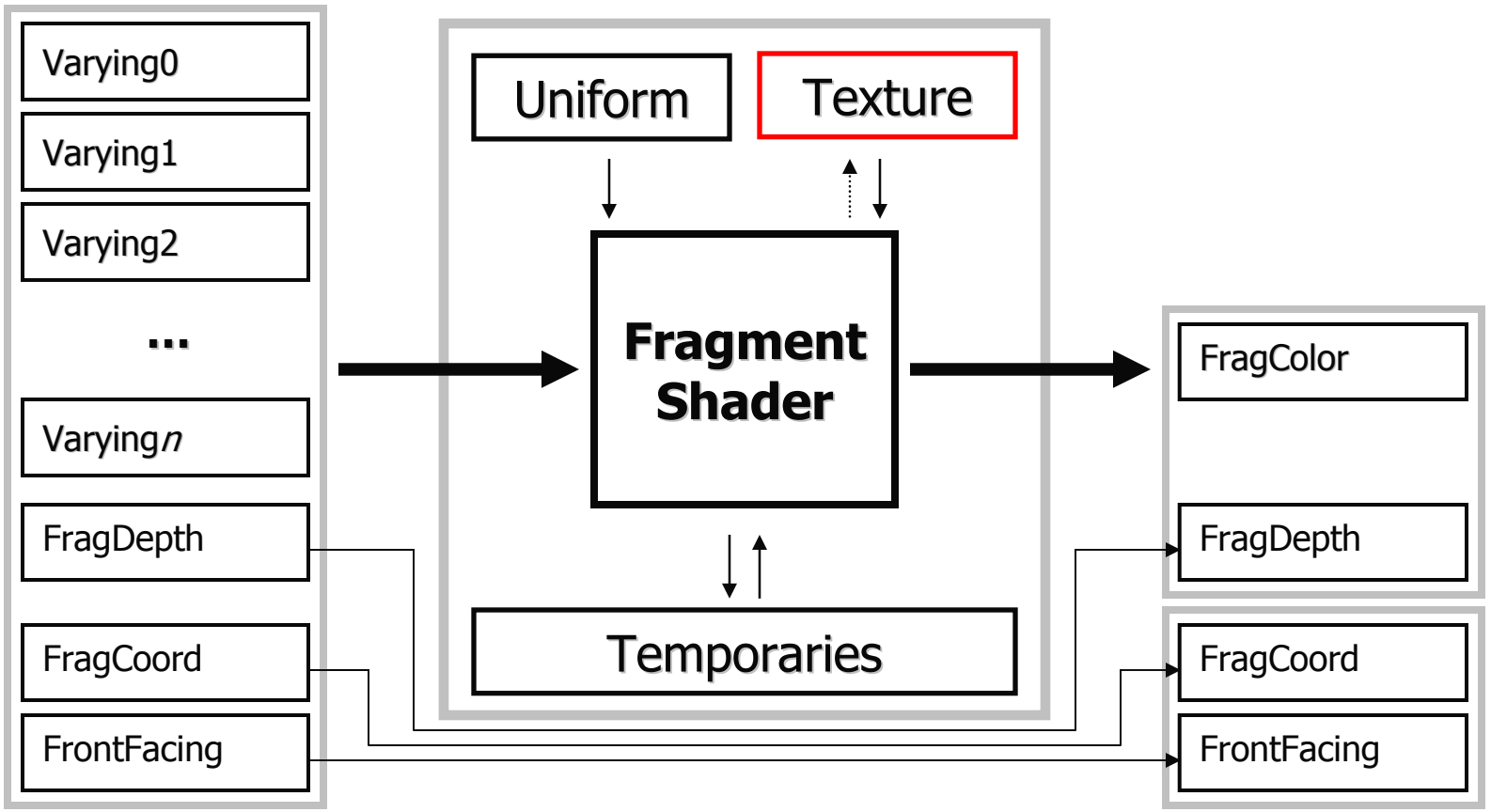
GL2 Fragment Limits Instructions (virtual)



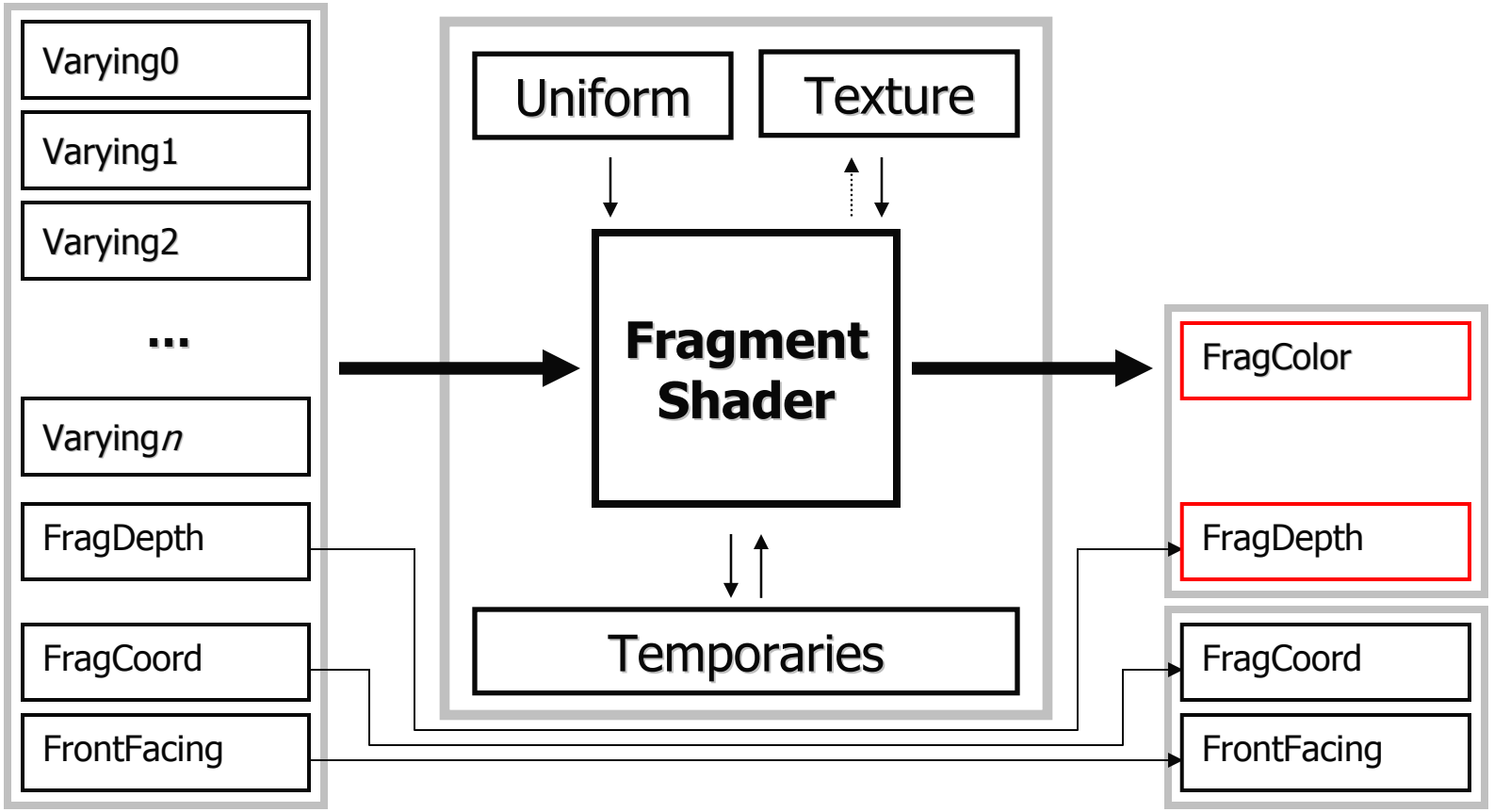
GL2 Fragment Limits Temporaries (virtual)



GL2 Fragment Limits
Texture Units (2)



GL2 Fragment Limits
Fragment Output (fixed)



GL2 Shading Language Today

- How we replace Fixed Function
- **Review Shading Language**
- Shading Language examples
- Procedural Shader demo

GL2 Shading Language Preprocessor directives

```
#                       #error
#define                 #pragma optimize(on|off)
#undef                  #pragma debug(on|off)

#if                     #line line file
#ifdef                  __LINE__
#ifndef                 __FILE__
#else                   __VERSION__
#elif
#endif                  // comment
                        /* comment */
```

GL2 Shading Language Types

- **void**
- **float vec2 vec3 vec4**
- **mat2 mat3 mat4**
- **int ivec2 ivec3 ivec4**
- **bool bvec2 bvec3 bvec4**
- **sampler*n*D samplerCube samplerShadow*n*D**

GL2 Shading Language Types

- **struct (with some minor restrictions)**
  - **No qualifiers**
  - **No bit fields**
  - **No forward references**
  - **No in-place definitions**
  - **No anonymous structures**

GL2 Shading Language Types

- arrays (with some minor restrictions)
  - one dimensional
  - size is integral constant expression
  - can declare unsized array, but…
  - specify size and type of array ONCE
  - any basic type and struct
  - NO initialization at declaration

GL2 Shading Language Reserved Types

- half hvec2 hvec3 hvec4
- fixed fvec2 fvec3 fvec4
- double dvec2 dvec3 dvec4
- sampler2DRect sampler3DRect

GL2 Shading Language Scope

- global
  - outside function
  - shared globals must be same type
- local (nested)
  - within function definition
  - within function
  - within compound statement

GL2 Shading Language Type Qualifiers

- const
- attribute
- uniform
- varying
- (default)

GL2 Shading Language Operators

- grouping: ()
- array subscript: []
- function call and constructor: ()
- field selector and swizzle: .
- postfix: ++ --
- prefix: ++ -- + - !

GL2 Shading Language Operators

- **binary: * / + -**
- **relational: < <= > >=**
- **equality: == !=**
- **logical: && ^^ ||**
- **selection: ?:**
- **assignment: = *= /= += -=**

GL2 Shading Language Reserved Operators

- **prefix: ~**
- **binary: %**
- **bitwise: << >> & ^ |**
- **assignment: %= <<= >>= &= ^= |=**

GL2 Shading Language Constructors

- **float()**
- **int()**
- **bool()**
- **vec2() vec3() vec4()**
- **mat2() mat3() mat4()**

GL2 Shading Language Scalar Constructors

```
float f; int i; bool b;

float( b)           // b=true?1.0:0.0;
int( b)             // b=true?1:0;
bool( b)            // identity
float( i)           // int to float
int( i)             // identity
bool( i)            // i!=0?true:false;
float( f)           // identity
int( f)             // float to int
bool( f)            // f!=0.0?true:false;
```

GL2 Shading Language Vector Constructors

```
vec2 v2; vec3 v3; vec4 v4;

vec2( 1.0 ,2.0)
vec3( 0.0 ,0.0 ,1.0)
vec4( 1.0 ,0.5 ,0.0 ,1.0)
vec4( 1.0)                    // all 1.0
vec4( v2 ,v2)                 // different...
vec4( v3 ,1.0)                // ...types
vec2( v4)
float( v4)
```

GL2 Shading Language Matrix Constructors

```
vec4 v4; mat4 m4;

mat4( 1.0, 2.0, 3.0, 4.0,
      5.0, 6.0, 7.0, 8.0,
      9.0, 10., 11., 12.,
      13., 14., 15., 16.)    // row major
mat4( v4, v4, v4, v4)
mat4( 1.0)                   // identity matrix
mat3( m4)                    // upper 3x3
vec4( m4)                    // 1st column
float( m4)                   // upper 1x1
```

GL2 Shading Language Structure Constructors

```
struct light {
  float intensity;
  vec3  position;
}
light headLight = light( 0.2
                         ,vec3( 0.0 ,0.0, 1.0));
// same as
light headLight;
headLight.intensity = 0.2;
headLight.position = vec3( 0.0 ,0.0, 1.0);
```

GL2 Shading Language Components

- component accessor for vectors
  - xyzw rgba stpq [i]
- component accessor for matrices
  - [i] [i][j]

GL2 Shading Language Vector components

```
vec2 v2;
vec3 v3;
vec4 v4;

v2.x    // is a float
v2.z    // wrong: component undefined for type
v4.rgba // is a vec4
v4.stp  // is a vec3
v4.b    // is a float
v4.xy   // is a vec2
v4.xgp  // wrong: mismatched component sets
```

GL2 Shading Language
Vector components (r-value)

```
vec2 v2;
vec3 v3;
vec4 v4;

v4.wzyx // swizzles, is a vec4
v4.bgra // swizzles, is a vec4
v4.xxxx // smears x, is a vec4
v4.xxx  // smears x, is a vec3
v4.yyxx // duplicates x and y, is a vec4
v2.yyyy // wrong: too many components for type
```

GL2 Shading Language Vector components (l-value)

```
vec4 v4 = vec4( 1.0, 2.0, 3.0, 4.0);

v4.xw = vec2( 5.0, 6.0); // (5.0, 2.0, 3.0, 6.0)
v4.wx = vec2( 7.0, 8.0); // (8.0, 2.0, 3.0, 7.0)
v4.xx = vec2( 9.0,10.0); // wrong: x used twice
v4.yz = 11.0;            // wrong: type mismatch
v4.yz = vec2( 12.0 );    // (8.0,12.0,12.0, 7.0)
```

GL2 Shading Language
Vector components (array)

```
vec4 v4 = vec4( 1.0, 2.0, 3.0, 4.0);
float f;
int i;

f = v4[0];                  // 1.0
f = v4[3];                  // 4.0
f = v4[4];                  // undefined
// ...
f = v4[i+1];                // defined for -1<i<3
```

GL2 Shading Language
Matrix components (array)

```
mat4 m4;

m4[1] = vec4(2.0);          // 2nd column = 2.0
m4[0][0] = 1.0;             // upper 1x1 = 1.0
m4[4][4] = 3.0;             // undefined
```

GL2 Shading Language Components (arrays)

```
vec3 v3[2];
mat3 m3[2];

v3[0]                   // is a vec3
v3[0][0]                // is a float
V3[0].x                 // is a float

m3[0]                   // is a mat3
m3[0][0]                // is a vec3, 1st column
m3[0][0][0]             // is a float, upper 1x1
                        // :-(
```

GL2 Shading Language Vector and Matrix Operations

```
mat4 m4,n4;
vec4 v4

v4 * m4                    // is a vec4
m4 * n4                    // is a mat4
```

GL2 Shading Language Flow-control (scalar)

- expression ? trueExpression : falseExpression
- if, if-else
- for, while, do-while
- return, break, continue
- discard (fragment only)

GL2 Shading Language User-defined functions

- Call by value-return
- in out inout keywords
- Function overloading
- One return value (can be any type)
- Scope rules same as C

GL2 Shading Language Built-in Vertex Special

```
vec4  gl_Position;      // must be written
vec4  gl_ClipPosition;  // may be written
float gl_PointSize;     // may be written
```

GL2 Shading Language Built-in Fragment Special

```
bool  gl_FrontFacing;    // may be read
float gl_FragColor;      // may be written
float gl_FragDepth;      // may be read/written
float gl_FragStencil;    // may be read/written
vec4  gl_FragDatan;      // may be written
vec4  gl_FragCoord;      // may be read
vec4  gl_FBColor;        // may be read
float gl_FBDepth;        // may be read
float gl_FBStencil;      // may be read
vec4  gl_FBDatan;        // may be read
```

GL2 Shading Language Built-in Attributes

```
// Vertex Attributes, p. 19
attribute vec4  gl_Vertex;
attribute vec3  gl_Normal;
attribute vec4  gl_Color;
attribute vec4  gl_SecondaryColor;
attribute vec4  gl_MultiTexCoordn;
attribute float gl_FogCoord;
```

GL2 Shading Language User-defined Attributes

```
attribute vec3  myTangent;
attribute vec3  myBinormal;
attribute float myTemperature;
attribute float myPressure;
attribute float myRainfall;
attribute float myTime;
// etc...
```

GL2 Shading Language Built-in Constants

```
// Implementation dependent constants
// Minimum Maximums
const int gl_MaxLights = 8;
const int gl_MaxClipPlanes = 6;
const int gl_MaxTextureUnits = 2;
const int gl_MaxTextureCoordsARB = 2;
// GL2 Minimum Maximums
const int gl_MaxVertexAttributesGL2 = 16;
const int gl_MaxVertexUniformWordsGL2 = 512;
const int gl_MaxVaryingFloatsGL2 = 32;
const int gl_MaxVertexTextureUnitsGL2 = 1;
const int gl_MaxFragmentTextureUnitsGL2 = 2;
const int gl_MaxFragmentUniformWordsGL2 = 64;
```

GL2 Shading Language Built-in uniforms

```
// Matrix state, p. 31, 32, 37, 39, 40
uniform   mat4  gl_ModelViewMatrix;
uniform   mat4  gl_ProjectionMatrix;
uniform   mat4  gl_ModelViewProjectionMatrix;
uniform   mat3  gl_NormalMatrix;
uniform   mat4  gl_TextureMatrix[n];
// Normal scaling, p. 33
uniform   float gl_NormalScale;
// Depth range in window coordinates, p. 33
struct gl_DepthRangeParameters {
  float near;                 // n
  float far;                  // f
  float diff;                 // f - n
};
uniform gl_DepthRangeParameters gl_DepthRange;
```

GL2 Shading Language Built-in uniforms

```
// Clip planes, p. 32
uniform   vec4  gl_ClipPlanes[gl_MaxClipPlanes]
// Point Size, p. 66, 67
struct gl_PointParameters {
  float size;
  float sizeMin;
  float sizeMax;
  float fadeThreshold;
  float distanceConstantAttenuation;
  float distanceLinearAttenuation;
  float distanceQuadraticAttenuation;
};
uniform gl_PointParameters gl_Point;
```

GL2 Shading Language Built-in uniforms

```
// Material State, p. 50, 55
struct gl_MaterialParameters {
  vec4  emission;
  vec4  ambient;
  vec4  diffuse;
  vec4  specular;
  float shininess;
};
uniform gl_MaterialParameters gl_FrontMaterial;
uniform gl_MaterialParameters gl_BackMaterial;
```

GL2 Shading Language Built-in uniforms

```
// Light State, p. 50, 53, 55
struct gl_LightSourceParameters {
  vec4  ambient;              // Acli
  vec4  diffuse;              // Dcli
  vec4  specular;             // Scli
  vec4  position;             // Ppli
  vec4  halfVector;           // Derived: Hi
  vec3  spotDirection;        // Sdli
  float spotExponent;         // Srli
  float spotCutoff;           // Crli  ([0.0,90.0],180.0)
  float spotCosCutoff;        // Derived: cos(Crli) ([1.0, 0.0],-1.0)
  float constantAttenuation   // K0
  float linearAttenuation     // K1
  float quadraticAttenuation // K2
};
uniform   gl_LightSourceParameters gl_LightSource[gl_MaxLights];
```

GL2 Shading Language Built-in uniforms

```
struct gl_LightModelParameters {
  vec4  ambient;              // Acs
};
uniform   gl_LightModelParameters gl_LightModel;
struct gl_LightModelProducts {
  vec4  sceneColor;           // Ecm + Acm * Acs
};
uniform   gl_LightModelProducts gl_FrontLightModelProduct;
uniform   gl_LightModelProducts gl_BackLightModelProduct;
struct gl_LightProducts {
  vec4  ambient;              // Acm * Acl
  vec4  diffuse;              // Dcm * Dcl
  vec4  specular;             // Scm * Scl
};
uniform   gl_LightProducts gl_FrontLightProduct[gl_MaxLights];
uniform   gl_LightProducts gl_BackLightProduct[gl_MaxLights];
```

GL2 Shading Language Built-in uniforms

```
// Texture Environment and Generation, p. 152, p. 40-42
uniform   vec4  gl_TextureEnvColor[gl_MaxFragmentTextureUnitsGL2];
uniform   vec4  gl_EyePlaneS[gl_MaxTextureCoordsARB]
                ,gl_EyePlaneT[gl_MaxTextureCoordsARB]
                ,gl_EyePlaneR[gl_MaxTextureCoordsARB]
                ,gl_EyePlaneQ[gl_MaxTextureCoordsARB];
uniform   vec4  gl_ObjectPlaneS[gl_MaxTextureCoordsARB]
                ,gl_ObjectPlaneT[gl_MaxTextureCoordsARB]
                ,gl_ObjectPlaneR[gl_MaxTextureCoordsARB]
                ,gl_ObjectPlaneQ[gl_MaxTextureCoordsARB];
```

GL2 Shading Language Built-in uniforms

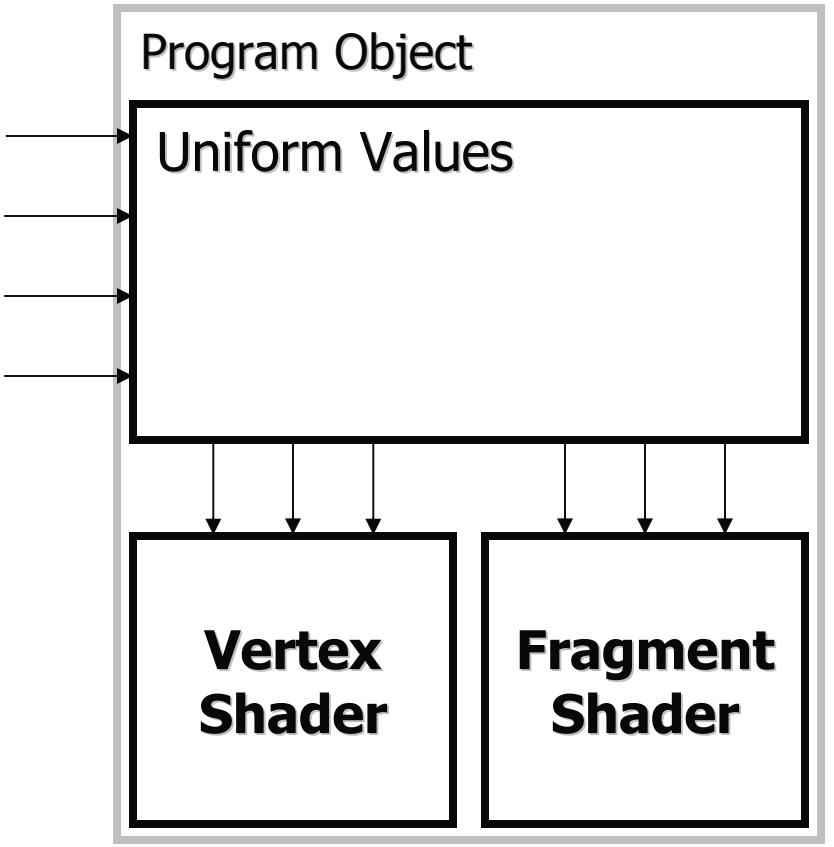
```
// Fog, p. 161
struct gl_FogParameters {
  vec4  color;
  float density;
  float start;
  float end;
  float scale;           // 1.0/(end-start)
};
uniform   gl_FogParameters gl_Fog;
```

GL2 Shading Language User-defined uniforms

```
uniform   mat4  myGlobalMatrix;
uniform   vec4  myAmbient;
uniform   vec4  myDiffuseMatLight;
uniform   vec4  mySpecularMatLight;
uniform   float myGlow;
uniform   vec4  myWarm;
uniform   vec4  myCool;
uniform   float myCurrentTime;
// etc...
```

GL2 Shading Language Uniforms

```
gl_ModelViewMatrix
gl_NormalMatrix;
myWarm;
myCurrentTime;
// etc...
```



GL2 Shading Language Built-in varying

```
varying   vec4  gl_FrontColor       // vertex
varying   vec4  gl_BackColor;       // vertex
varying   vec4  gl_FrontSecColor;   // vertex
varying   vec4  gl_BackSecColor;    // vertex

varying   vec4  gl_Color;           // fragment
varying   vec4  gl_SecondaryColor;  // fragment

varying   vec4  gl_TexCoord[];      // both
varying   float gl_FogFragCoord;    // both
```

GL2 Shading Language
User-defined varying

```
varying  vec3  myNormalPrime;
varying  vec3  myTangentPrime;
varying  vec3  myBinormalPrime;
varying  float myPressurePrime;
// etc...
```

GL2 Built-in Functions
Angles and Trigonometry

```
genType radians( genType degrees)
genType degrees( genType radians)
genType sin( genType angle)          // radians
genType cos( genType angle)
genType tan( genType angle)
genType asin( genType angle)
genType acos( genType angle)
genType atan( genType y ,genType x)
genType atan( genType y_over_x)
```

GL2 Built-in Functions Exponentials, etc.

```
genType pow( genType x ,genType y)
genType exp2( genType x)
genType log2( genType x)
genType sqrt( genType x)
genType inversesqrt( genType x)
```

GL2 Built-in Functions Common

```
genType abs( genType x)
genType sign( genType x)
genType floor( genType x)
genType ceil( genType x)
genType fract( genType x)
```

GL2 Built-in Functions Common

```
genType mod( genType x ,float y)
genType mod( genType x ,genType y)
genType min( genType x ,genType y)
genType max( genType x ,genType y)
genType clamp( genType x
               ,float v0, float v1)
genType clamp( genType x
               ,genType v0, genType v1)
```

GL2 Built-in Functions Interpolations

```
// return x*( 1.0-a) + y*a
genType mix( genType x ,genType y ,float a)
genType mix( genType x ,genType y ,genType a)
// x <= edge ? 0.0 : 1.0
genType step( float edge ,genType x)
genType step( genType edge ,genType x)
```

GL2 Built-in Functions Interpolations

```
// genType t;
// t = (x-edge0)/(edge1-edge0);
// t = clamp( t, 0.0, 1.0);
// return t*t*(3.0-2.0*t);
genType smoothstep( float edge0
                   ,float edge1
                   ,genType x)
genType smoothstep( genType edge0
                   ,genType edge1
                   ,genType x)
```

GL2 Built-in Functions Geometric

```
float   length( genType x)
float   distance( genType p0 ,genType p1)
float   dot( genType x ,genType y)
vec3    cross( vec3 x ,vec3 y)
genType normalize( genType x)
genType faceforward( genType N
                     ,genType I
                     ,genType Nref)
genType reflect( genType I
                 ,genType N)
```

GL2 Built-in Functions
Matrix

```
mat      matrixCompMult( mat x ,mat y)
```

GL2 Built-in Functions Vector Relational

```
bGenType lessThan( genType x ,genType y)
bGenType lessThanEqual( genType x
                        ,genType y)
bGenType greaterThan( genType x ,genType y)
bGenType greaterThanEqual( genType x
                           ,genType y)
bGenType equal( genType x ,genType y)
bGenType notEqual( genType x , genType y)
bGenType not( genType x)
bool any( bGenType x)
bool all( bGenType x)
```

GL2 Built-in Functions Texture

```
vec4    texture1D( sampler1D sampler
                   ,float coord ,float bias)
vec4    texture2D( sampler2D sampler
                   ,vec2 coord ,float bias)
vec4    texture3D( sampler3D sampler
                   ,vec3 coord ,float bias)
vec4    textureCube( samplerCube sampler
                     ,vec3 coord ,float bias)
```

GL2 Built-in Functions Texture

```
vec4    texture1DProj( sampler1D sampler
                       ,vec2 coord ,float bias)
vec4    texture2DProj( sampler2D sampler
                       ,vec3 coord ,float bias)
vec4    texture3DProj( sampler3D sampler
                       ,vec4 coord ,float bias)
vec4    textureCubeProj( samplerCube sampler
                         ,vec4 coord ,float bias)
```

GL2 Built-in Functions Texture

```
vec4    texture1DProj( sampler1D sampler
                       ,vec4 coord ,float bias)
vec4    texture2DProj( sampler2D sampler
                       ,vec4 coord ,float bias)
```

GL2 Built-in Functions Shadow

```
vec4    shadow1D( samplerShadow1D sampler
                  ,vec3 coord ,float bias)
vec4    shadow2D( samplerShadow2D sampler
                  ,vec3 coord ,float bias)
vec4    shadow1DProj( samplerShadow1D sampler
                      ,vec4 coord ,float bias)
vec4    shadow2DProj( samplerShadow2D sampler
                      ,vec4 coord ,float bias)
```

GL2 Built-in Functions Vertex

```
vec4 ftransform()

// Example usage:
gl_Position = ftransform();
```

GL2 Built-in Functions Fragment

```
genType dFdx( genType p)
genType dFdy( genType p)
genType fwidth( genType p)
```

GL2 Built-in Functions
Noise

```
float   noise1( genType coord)
vec2    noise2( genType coord)
vec3    noise3( genType coord)
vec4    noise4( genType coord)
```

GL2 Shading Language Today

- How we replace Fixed Function
- Review Shading Language
- **Shading Language examples**
- Procedural Shader demo

GL2 Trivial Example Vertex Shader

```
varying vec3 Normal;          // output
#define MVP    gl_ModelViewProjectionMatrix
#define MV     gl_ModelViewMatrix
#define MV_IT  gl_NormalMatrix
void main( void)              // Vertex Shader
{
  gl_Position = MVP * gl_Vertex;
//gl_ClipVertex = MV * gl_Vertex;
  Normal = normalize( MV_IT * gl_Normal);
}
```

GL2 Trivial Example Fragment Shader

```
varying vec3 Normal;          // input
uniform vec3 LightColor;      // Light Color
uniform vec3 LightPos;        // Light Position

void main( void)              // Fragment Shader
{
  vec3 color = LightColor;
  color *=
    max(0.0,dot(normalize(Normal),LightPos));
  gl_FragColor = vec4( color,1.0);
}
```

GL2 ToyBall Example Vertex Shader

```
varying vec4  v_ObjPos;
varying vec4  v_EyePos;

void main( void)                        // Vertex Shader
{
   vec4 ObjPos;
   vec4 EyePos

   ObjPos = gl_Vertex;
   gl_Position = gl_ModelViewProjectionMatrix * ObjPos;
   v_ObjPos = ObjPos;

   EyePos = gl_ModelViewMatrix * ObjPos;
   v_EyePos = EyePos;
}
```

GL2 ToyBall Example Fragment Shader

```
varying vec4  v_EyePos;
varying vec4  v_ObjPos;

uniform vec4  myLightDir;          // NOTA BENE:  please normalize
uniform vec4  myEyeDir;            // (0.0, 0.0, 1.0, 0.0)
uniform vec4  myHVector;
uniform vec4  myEyeCenter;
```

GL2 ToyBall Example Fragment Shader

```
uniform vec4  mySpecularColor;
uniform vec4  myRed, myYellow, myBlue;

uniform vec4  myHalfSpace0;
uniform vec4  myHalfSpace1;
uniform vec4  myHalfSpace2;
uniform vec4  myHalfSpace3;
uniform vec4  myHalfSpace4;

uniform vec4  toyBallConst;        // (-3.0, 0.4, 0.0, 0.0)
uniform vec4  one;
uniform float myFWidth;
```

GL2 ToyBall Example Fragment Shader

```
void main( void)                        // Fragment Shader
{
   vec4  NNormal;
   vec4  P;                             // The point in shader space
   vec4  SurfColor;
   float Intensity;

   vec4  PDistance;
   float InOrOut;

   P = v_ObjPos;                        // Analytically calculate P
   P.w = 0.0;
   P = normalize( P);
   P.w = 1.0;
```

GL2 ToyBall Example Fragment Shader

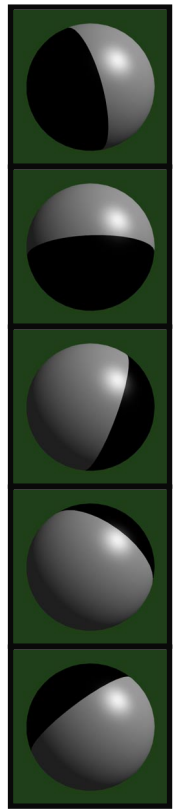
```
InOrOut = toyBallConst.x;

PDistance.x = dot( P, myHalfSpace0);                    // Fig 1
PDistance.y = dot( P, myHalfSpace1);                    // Fig 2
PDistance.z = dot( P, myHalfSpace2);                    // Fig 3
PDistance.w = dot( P, myHalfSpace3);                    // Fig 4

PDistance = smoothstep( -myFWidth, myFWidth, PDistance);

InOrOut += dot( PDistance, one);

PDistance.x = dot( P, myHalfSpace4);                    // Fig 5
PDistance.y = toyBallConst.y - abs( P.z);
PDistance = smoothstep( -myFWidth, myFWidth, PDistance);
```



GL2 ToyBall Example Fragment Shader

```
InOrOut += PDistance.x;                                  // Fig 1

InOrOut = clamp( InOrOut, 0.0, 1.0);                     // Fig 2

SurfColor = mix( myYellow, myRed, InOrOut);              // Fig 3

SurfColor = mix( SurfColor, myBlue, PDistance.y );       // Fig 4
```

GL2 ToyBall Example Fragment Shader

```
//
// Analytically calculate the sphere NNormal in eye coordinates
   NNormal = v_EyePos - myEyeCenter;
   NNormal = normalize( NNormal);
//
// Per fragment diffuse lighting
   Intensity = 0.2; // ambient
   Intensity += clamp( dot( myLightDir, NNormal), 0.0, 1.0) * 0.8;

   SurfColor *= Intensity;                                    // Fig 1
```

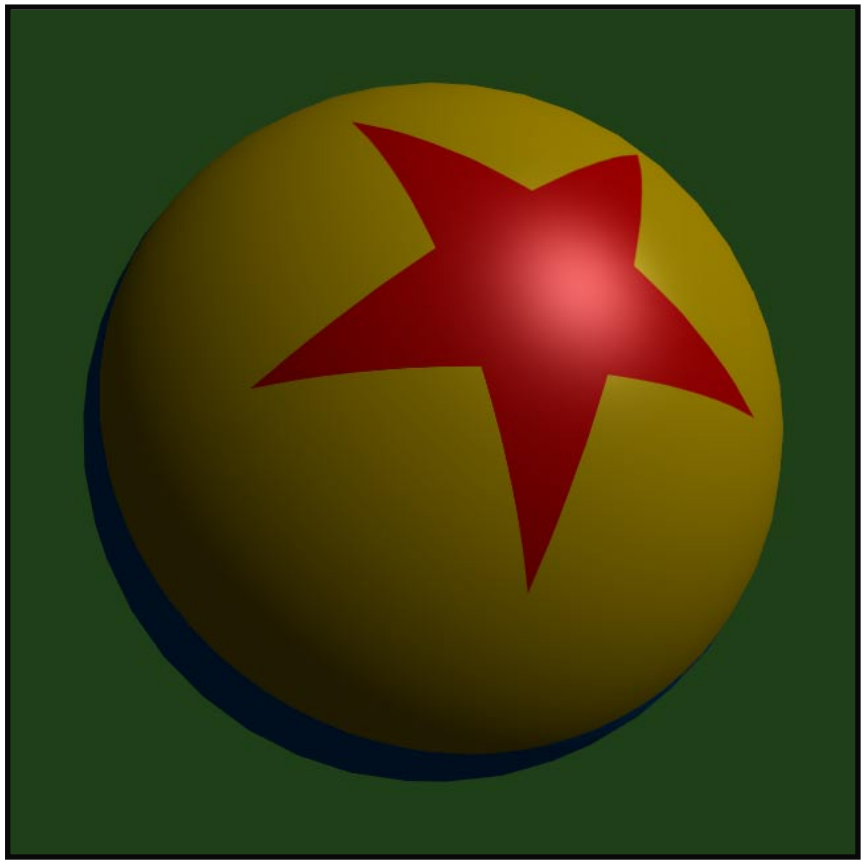
GL2 ToyBall Example Fragment Shader

```
//
// Per fragment specular lighting
    Intensity = clamp( dot( myHVector, NNormal), 0.0, 1.0);
    Intensity = pow ( Intensity, mySpecularColor.a);

    SurfColor += mySpecularColor * Intensity;            // Fig 1

    SurfColor.a = 1.0;
    gl_FragColor = SurfColor;
}
```

GL2 ToyBall Demo



GL2 Shading Language Questions?

- How we replace Fixed Function
- Review Shading Language
- Trivial Shading Language example
- Procedural Shader demo

- bill@ati.com