



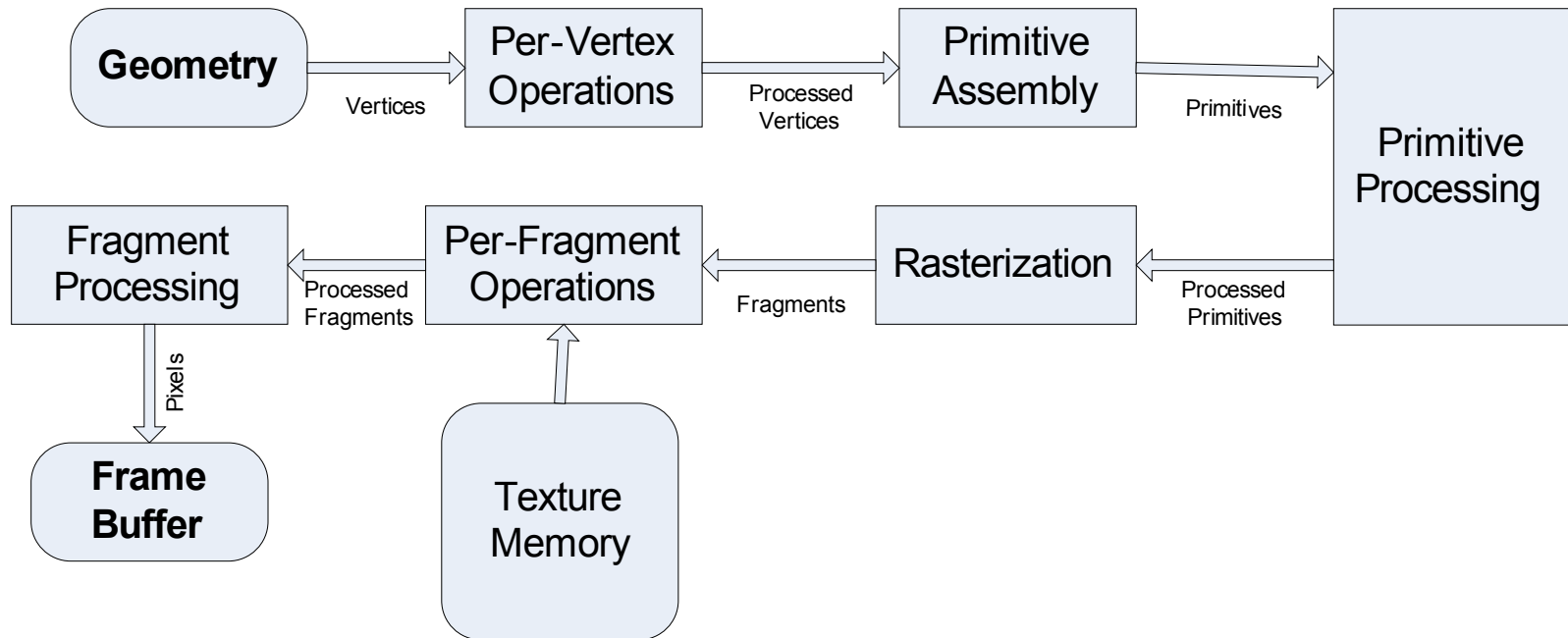
# Shaders en OpenGL



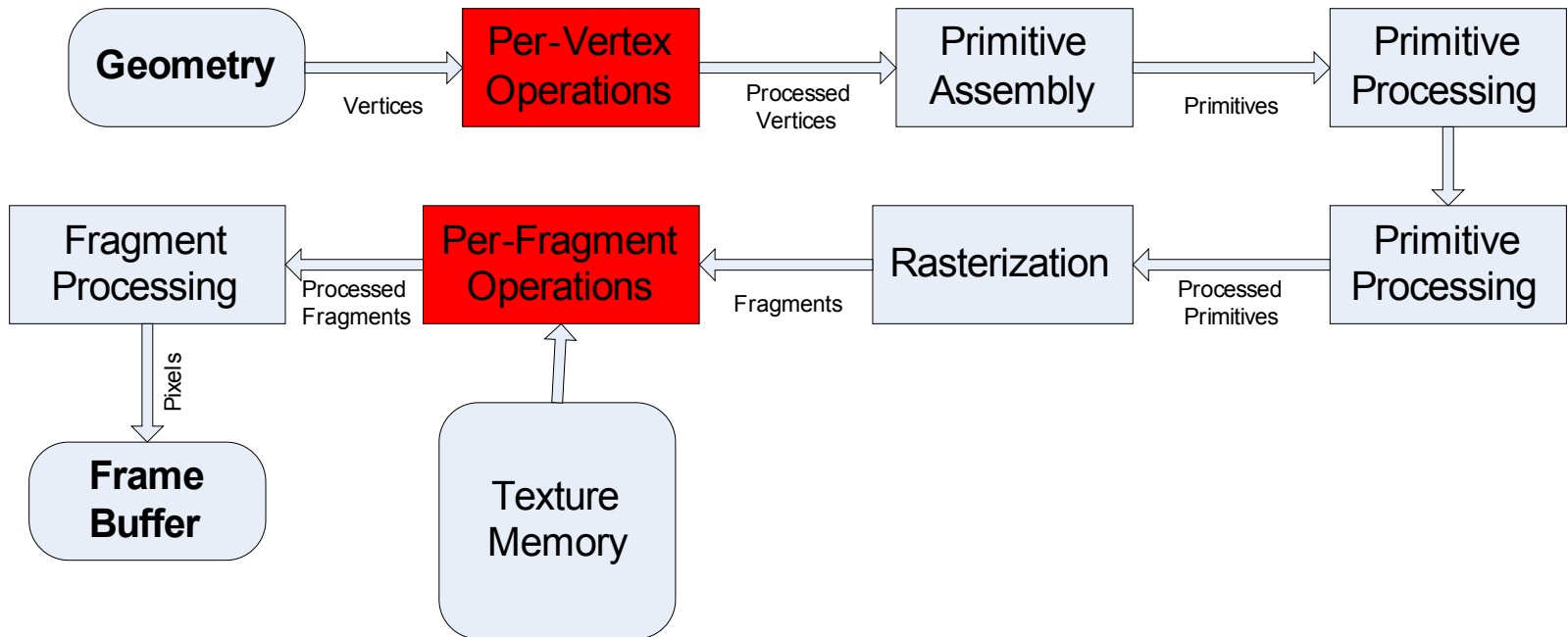
# Introducción a Shaders en OpenGL

- vertex shaders, fragment shaders, y how como se comunican
- Built-in datatypes y funciones
- Un shader ejemplo que implementa bump mapping
- Como compilar, inicializar y activar un shader

# OpenGL 1.5



# OpenGL 2.0



# OpenGL Shading Language (GLSL)

Lenguaje C-like, tiene sintaxis y flujo de control similares

- Más restringido en áreas
  - Funciones: retorno por valor, paso por valor
- Variables built-in especiales para entrada\salida. Comienzan con “gl\_”
- Tipos de datos built-in para matrices y vectores (2D, 3D y 4D)
- Funciones built-in utilitarias: dot, cos, sin, mix, ...

# Swizzling (acceso)

El selector de estructuras (.) ( [SWIZZLE](#) ) se usa para acceder los componentes de un vector o reordenarlos, listando los nombres luego del operador (.)

- `vec4 v4;`

`v4.rgba;` // es un `vec4` y lo mismo que usar sólo `v4`,

`v4.rgb;` // es un `vec3`

`v4.b;` // es un `float`,

`v4.xy;` // es un `vec2`,

`v4.xgba;` // es ilegal - los componentes no son del mismo conjunto.

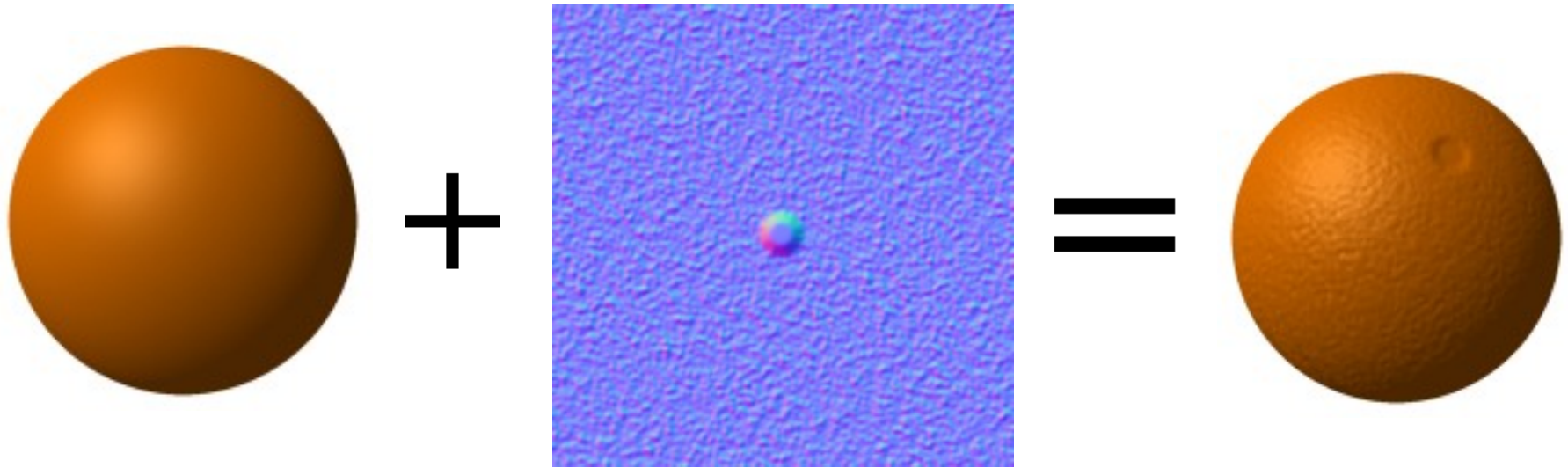
- Los nombres de componentes pueden usarse para reordenar los mismos y replicar en otro conjunto:

- `v3 = v4.zxy` // es igual a `v3.xyz = v4.zyx`, permuta coordenadas

# Tres tipos de variables

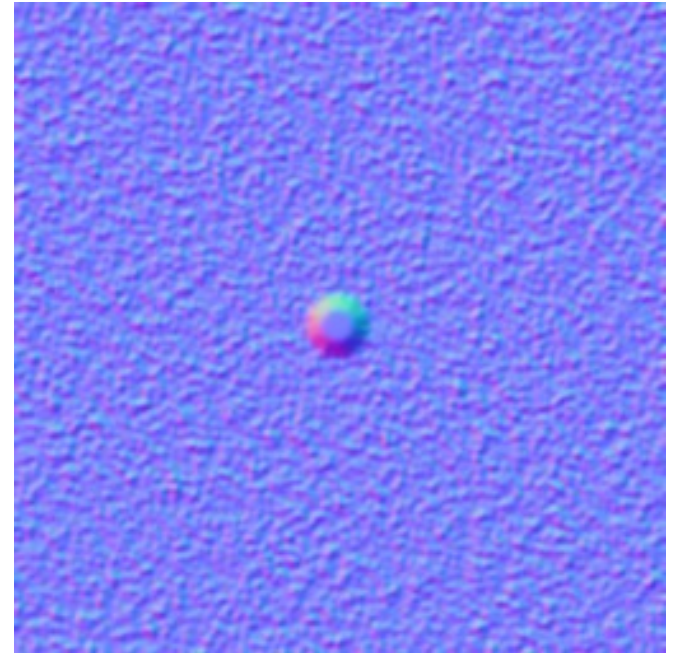
- **Uniform:** pueden ser mutadas una sólo vez por primitiva
  - uniform bool bumpOn;
- **Attribute:** pueden ser modificadas en cualquier momento, representan atributos asociados con vertices
  - attribute vec3 tangentVec;
- **Varying:** comunican entre el vertex shader y el fragment shader. Son interpolados en los polígonos.
  - varying vec3 lightVec;

# Normal Mapping

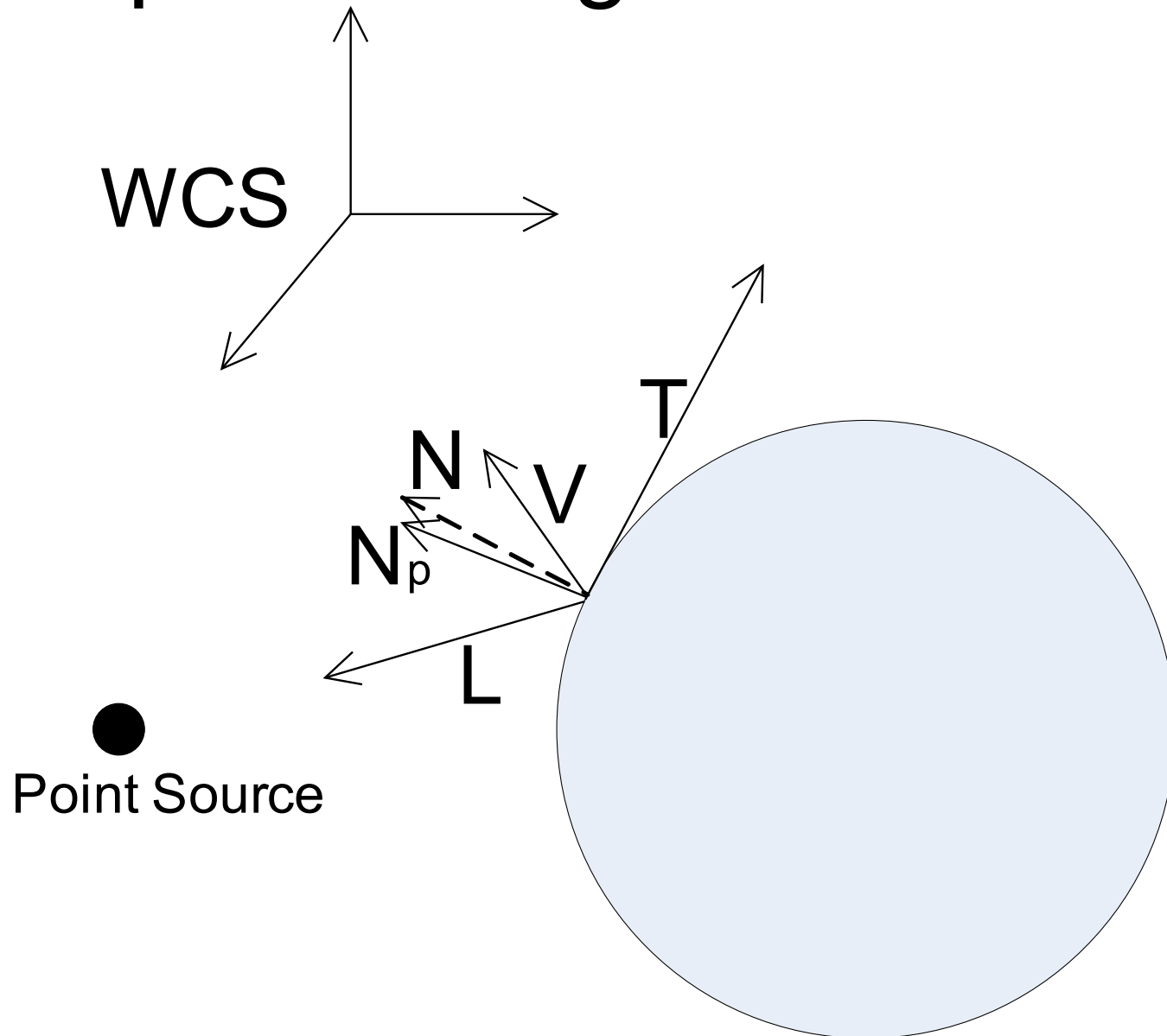


# Normal Maps y Tangent Maps

- Una imagen donde cada pixel representa una normal:
  - R -> x, G -> y, B -> z
- Cada normal es perturbada una pequeña cantidad, así que normal/tangent maps tienden a ser “azulados” en color
- R, G, y B van de 0.0 a 1.0
- $N.x = ( R - 0.5 ) * 2.0, \dots$



# El Espacio Tangente



# Normal Mapping Vertex Shader

```
1.     uniform bool bumpon;
2.     attribute vec3 tangentVec;
3.     varying vec3 lightVec;
4.     varying vec3 eyeVec;
5.
6.     void main (void)
7.     {
8.         vec4 ecPosition = gl_ModelViewMatrix * gl_Vertex;
9.
10.        gl_Position = ftransform();
11.        gl_TexCoord[0] = gl_MultiTexCoord0;//texture mapping stuff
12.
13.        vec3 orgLightVec = ( gl_LightSource[0].position.xyz - ecPosition.xyz);
14.
15.        vec3 n = normalize(gl_NormalMatrix * gl_Normal);
16.        vec3 t = normalize(gl_NormalMatrix * tangentVec);
17.        vec3 b = cross(n, t);
18.
19.        lightVec.x = dot( orgLightVec, t);
20.        lightVec.y = dot( orgLightVec, b);
21.        lightVec.z = dot( orgLightVec, n);
22.
23.        vec3 position = -ecPosition.xyz;
24.        eyeVec.x = dot( position, t);
25.        eyeVec.y = dot( position, b);
26.        eyeVec.z = dot( position, n);
27.    }
```

# Normal Mapping Fragment Shader

```
1.     uniform bool bumpon;
2.     uniform sampler2D normalMap;//must initialize with texture unit integer
3.     varying vec3 lightVec;
4.     varying vec3 eyeVec;
5.
6.     void main (void)
7.     {
8.         vec3 N = vec3(0.0, 0.0, 1.0);
9.         if( bumpon ) N = normalize( ( texture2D(normalMap, gl_TexCoord[0].xy).xyz - 0.5) * 2.0 );
10.
11.         vec3 L = normalize(lightVec);
12.         vec3 V = normalize(eyeVec);
13.         vec3 R = reflect(L, N);
14.
15.         float pf = pow( dot(R, V), gl_FrontMaterial.shininess );
16.
17.         vec4 GlobalAmbient = gl_LightModel.ambient;
18.         vec4 Ambient = gl_LightSource[0].ambient * gl_FrontMaterial.ambient;
19.         vec4 Diffuse = gl_LightSource[0].diffuse * gl_FrontMaterial.diffuse * dot(N, L);
20.         vec4 Specular = gl_LightSource[0].specular * gl_FrontMaterial.specular * pf;
21.
22.         vec4 color1 = GlobalAmbient + Ambient + Diffuse;
23.         vec4 color2 = Specular;
24.
25.         gl_FragColor = clamp( color1 + color2, 0.0, 1.0 );
26.     }
```

# Compilation

```
char* vertSrc = readShaderToString(vertShader);  
char* fragSrc = readShaderToString(fragShader);
```

```
vs = glCreateShader(GL_VERTEX_SHADER);  
fs = glCreateShader(GL_FRAGMENT_SHADER);
```

```
glShaderSource(vs, 1, &vertSrc, NULL);  
glShaderSource(fs, 1, &fragSrc, NULL);
```

```
glCompileShader(vs);  
printOpenGLError(); // Check for OpenGL errors  
glGetShaderiv(vs, GL_COMPILE_STATUS, &vertCompiled);  
printShaderInfoLog(vs);
```

```
glCompileShader(fs);  
printOpenGLError(); // Check for OpenGL errors  
glGetShaderiv(fs, GL_COMPILE_STATUS, &fragCompiled);  
printShaderInfoLog(fs);
```

```
if (!vertCompiled || !fragCompiled) exit(-1);
```



# Compilation

```
// Create a program object and attach the two compiled shaders
```

```
int program = glCreateProgram();  
glAttachShader(program, vs);  
glAttachShader(program, fs);
```

```
// Link the program object and print out the info log
```

```
glLinkProgram(program);  
printOpenGLError(); // Check for OpenGL errors  
glGetProgramiv(program, GL_LINK_STATUS, &linked);  
printProgramInfoLog();
```

```
if (!linked) exit(-1)
```

# Initialization and Activation

- Call **glUseProgram( program )** to activate a shader.
  - The default functionality will be disabled
  - Can switch between multiple shaders while rendering a frame
  - `glUseProgram(0)` to turn on default pipeline

- Initialization example:

```
GLint loc = glGetUniformLocation(program, "bumpon");
glUniform1i( loc, GL_TRUE );
GLint loc = glGetUniformLocation(program, "normalMap");
glUniform1i( loc, 0 );
GLint loc = glGetUniformLocation(program, "tangentVec");
glVertexAttrib3f(loc, v1, v2, v3);
```

# Class shaderProgram

```
class shaderProgram {
public:
    enum shaderT{
        NORMAL,
        NONE
    };
    static const unsigned NUM_OF_SHADERS = 2;

    void initShaders() {
        init(NORMAL, "shaders/normal.vert", "shaders/normal.frag");
    }

    inline void setActive(shaderT p);
    inline void uniform1i(char *var, GLint val);
    ...
};
```



# Usage

```
#include "shader.hpp"
```

```
void main(){  
    //called after OpenGL rendering context has been initialized  
    g_shader.initShaders();  
    ...  
    g_shader.setActive(NORMAL);  
    g_shader.Uniform1i("bumpon", GL_TRUE);  
    ...  
}
```



For more information, see the spec

- <http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf>