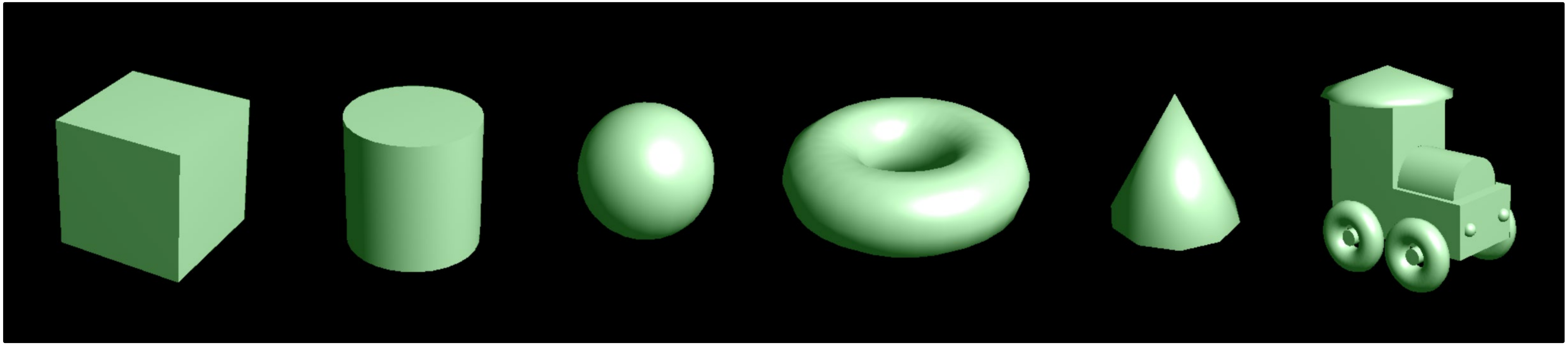
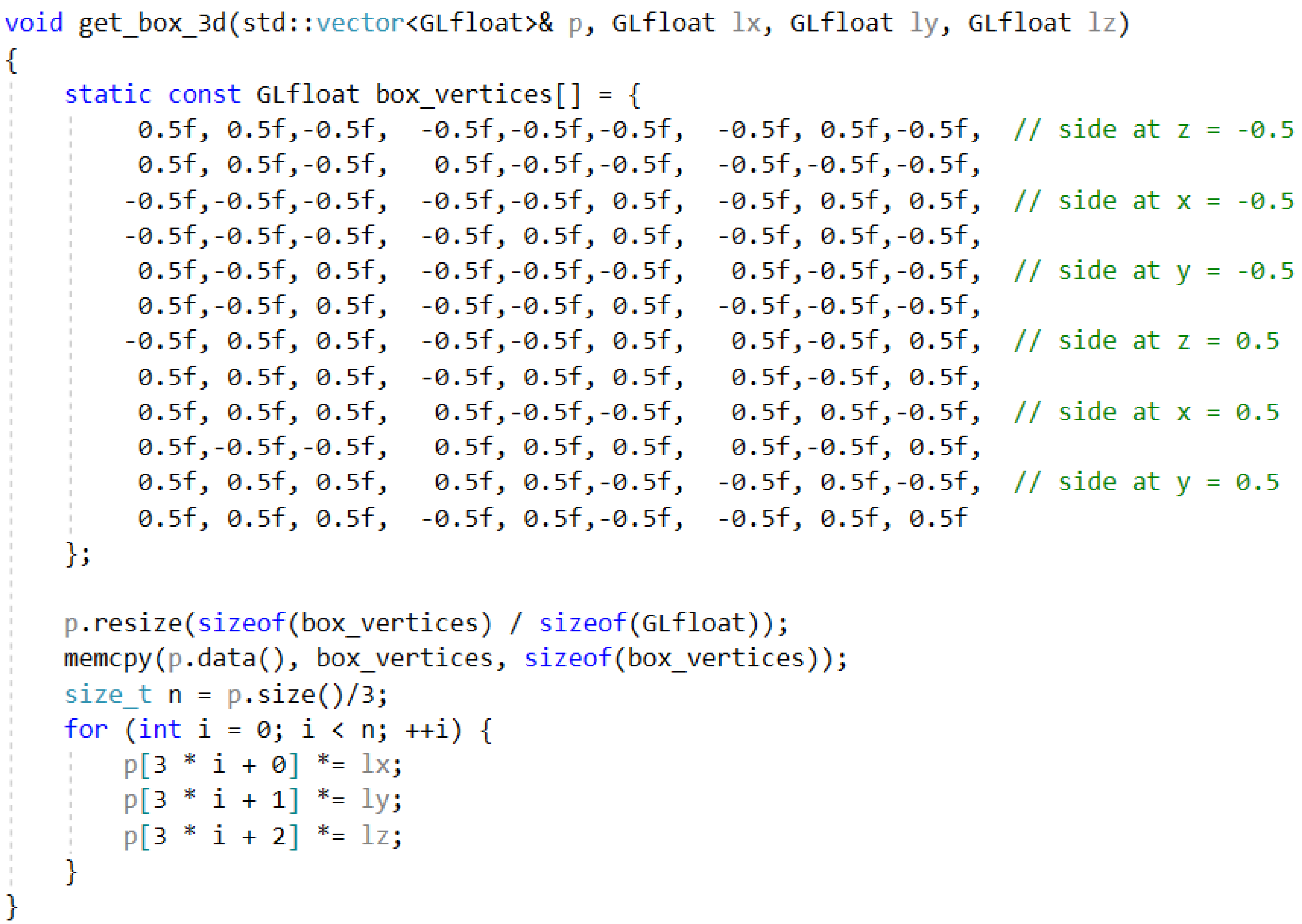
Homework 10: Shading Primitives

- Modify the previous homework solution to render the primitives with Gouraud shading and Phong shading according to user input.
- Requirements
  - Key D (capital D): Original Color Mode
  - Key P (capital P): Phong Shading Mode
  - Key G (capital G): Gouraud Shaing Mode



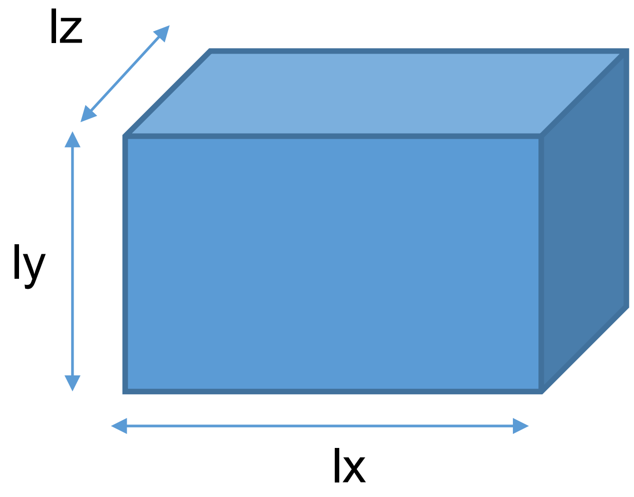
Computing normal vectors

- **Box primitive**



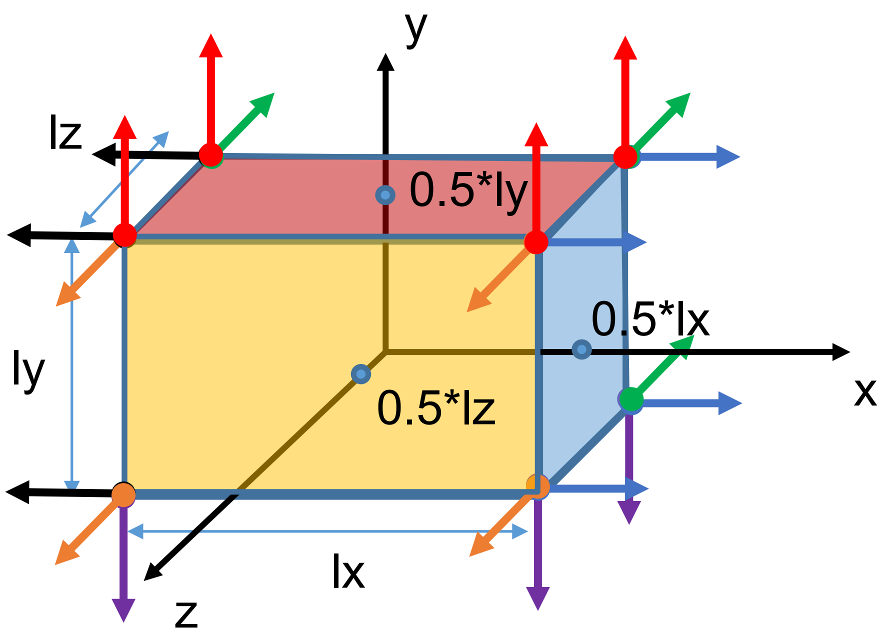
```
void get_box_3d(std::vector<GLfloat>& p, GLfloat lx, GLfloat ly, GLfloat lz)
{
    static const GLfloat box_vertices[] = {
         0.5f, 0.5f,-0.5f,  -0.5f,-0.5f,-0.5f,  -0.5f, 0.5f,-0.5f,  // side at z = -0.5
         0.5f, 0.5f,-0.5f,   0.5f,-0.5f,-0.5f,  -0.5f,-0.5f,-0.5f,
        -0.5f,-0.5f,-0.5f,  -0.5f,-0.5f, 0.5f,  -0.5f, 0.5f, 0.5f,  // side at x = -0.5
        -0.5f,-0.5f,-0.5f,  -0.5f, 0.5f, 0.5f,  -0.5f, 0.5f,-0.5f,
         0.5f,-0.5f, 0.5f,  -0.5f,-0.5f,-0.5f,   0.5f,-0.5f,-0.5f,  // side at y = -0.5
         0.5f,-0.5f, 0.5f,  -0.5f,-0.5f, 0.5f,  -0.5f,-0.5f,-0.5f,
        -0.5f, 0.5f, 0.5f,  -0.5f,-0.5f, 0.5f,   0.5f,-0.5f, 0.5f,  // side at z = 0.5
         0.5f, 0.5f, 0.5f,  -0.5f, 0.5f, 0.5f,   0.5f,-0.5f, 0.5f,
         0.5f, 0.5f, 0.5f,   0.5f,-0.5f,-0.5f,   0.5f, 0.5f,-0.5f,  // side at x = 0.5
         0.5f,-0.5f,-0.5f,   0.5f, 0.5f, 0.5f,   0.5f,-0.5f, 0.5f,
         0.5f, 0.5f, 0.5f,   0.5f, 0.5f,-0.5f,  -0.5f, 0.5f,-0.5f,  // side at y = 0.5
         0.5f, 0.5f, 0.5f,  -0.5f, 0.5f,-0.5f,  -0.5f, 0.5f, 0.5f
    };

    p.resize(sizeof(box_vertices) / sizeof(GLfloat));
    memcpy(p.data(), box_vertices, sizeof(box_vertices));
    size_t n = p.size()/3;
    for (int i = 0; i < n; ++i) {
        p[3 * i + 0] *= lx;
        p[3 * i + 1] *= ly;
        p[3 * i + 2] *= lz;
    }
}
```

Existing code



- Modified code

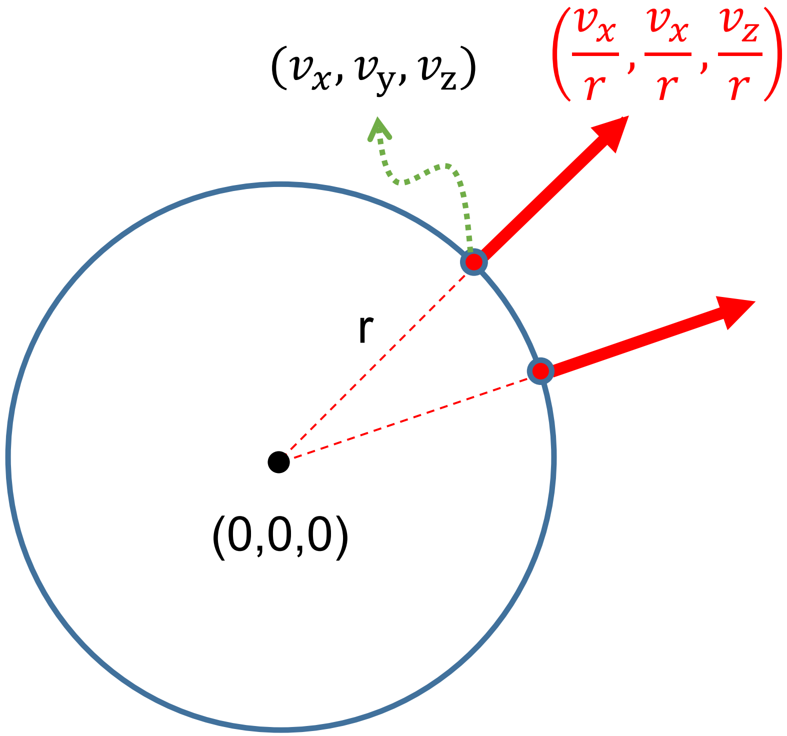
```
void get_box_3d(
    std::vector<GLfloat>& p,
    std::vector<GLfloat>& normals,
    GLfloat lx,
    GLfloat ly,
    GLfloat lz)
{
    // ... Existing implementation ...

    // Compute normals
    normals.resize(n * 3);
    float* cursor = normals.data();

    // For vertices on the side at z = -0.5
    for (int i = 0; i < 6; ++i, cursor += 3) { cursor[0] = 0;   cursor[1] = 0;  cursor[2] = -1; }
    // For vertices on the side at x = -0.5
    for (int i = 0; i < 6; ++i, cursor += 3) { cursor[0] = -1;  cursor[1] = 0;  cursor[2] = 0; }
    // For vertices on the side at y = -0.5
    for (int i = 0; i < 6; ++i, cursor += 3) { cursor[0] = 0;   cursor[1] = -1; cursor[2] = 0; }
    // For vertices on the side at z = 0.5
    for (int i = 0; i < 6; ++i, cursor += 3) { cursor[0] = 0;   cursor[1] = 0;  cursor[2] = 1; }
    // For vertices on the side at x = 0.5
    for (int i = 0; i < 6; ++i, cursor += 3) { cursor[0] = 1;   cursor[1] = 0;  cursor[2] = 0; }
    // For vertices on the side at y = 0.5
    for (int i = 0; i < 6; ++i, cursor += 3) { cursor[0] = 0;   cursor[1] = 1;  cursor[2] = 0; }
}
```



- How to upload the normal data in OpenGL:

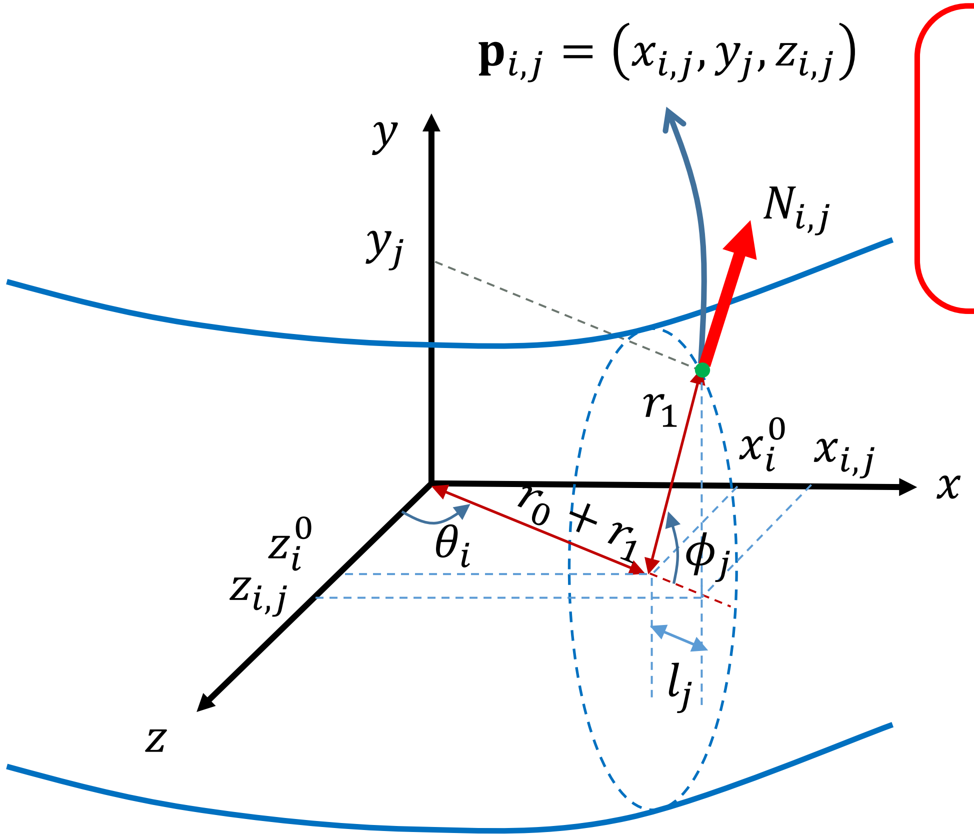
```
GLchar* attri_name[3] = { "vPosition", "vNormal", "vColor" };
GLvec* vtx_list[3] = { &vtx_pos, &vtx_nml, &vtx_clrs };

glGenBuffers(3, buffs);
for (int i = 0; i < 3; ++i) {
    glBindBuffer(GL_ARRAY_BUFFER, buffs[i]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(GLfloat)*vtx_list[i]->size(), vtx_list[i]->data(), GL_STATIC_DRAW);
    GLint location = glGetAttribLocation(program, attri_name[i]);
    glVertexAttribPointer(location, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(location);
}
```

- **Sphere Primitive**



```
void get_sphere_3d(std::vector<GLfloat>& p,
     std::vector<GLfloat>& normals,
     GLfloat r, GLint subh, GLint suba)
{
     ......
     if (i < subh) {
         // first triangle (v0 - v1 - v3)
         FPUSH_VTX3(p, vx0, vy0, vz0);
         FPUSH_VTX3(p, vx1, vy1, vz1);
         FPUSH_VTX3(p, vx3, vy3, vz3);
         FPUSH_VTX3(normals, vx0/r, vy0/r, vz0/r);
         FPUSH_VTX3(normals, vx1/r, vy1/r, vz1/r);
         FPUSH_VTX3(normals, vx3/r, vy3/r, vz3/r);
     }

     if (1 < i) {
         // second triangle (v3 - v2 - v0)
         FPUSH_VTX3(p, vx3, vy3, vz3);
         FPUSH_VTX3(p, vx2, vy2, vz2);
         FPUSH_VTX3(p, vx0, vy0, vz0);
         FPUSH_VTX3(normals, vx3/r, vy3/r, vz3/r);
         FPUSH_VTX3(normals, vx2/r, vy2/r, vz2/r);
         FPUSH_VTX3(normals, vx0/r, vy0/r, vz0/r);
     }
     ......
}
```

- **Torus Primitive**



$$\boxed{N_{i,j} = \frac{1}{r_1}\left(\mathbf{p}_{i,j} - \begin{bmatrix} x_i^0 \\ 0 \\ z_i^0 \end{bmatrix}\right) = \frac{1}{r_1}\left(\begin{bmatrix} x_i^0 + \Delta x_{i,j} \\ y_j \\ z_i^0 + \Delta z_{i,j} \end{bmatrix} - \begin{bmatrix} x_i^0 \\ 0 \\ z_i^0 \end{bmatrix}\right) = \frac{1}{r_1}\begin{bmatrix} \Delta x_{i,j} \\ y_j \\ \Delta z_{i,j} \end{bmatrix}}$$

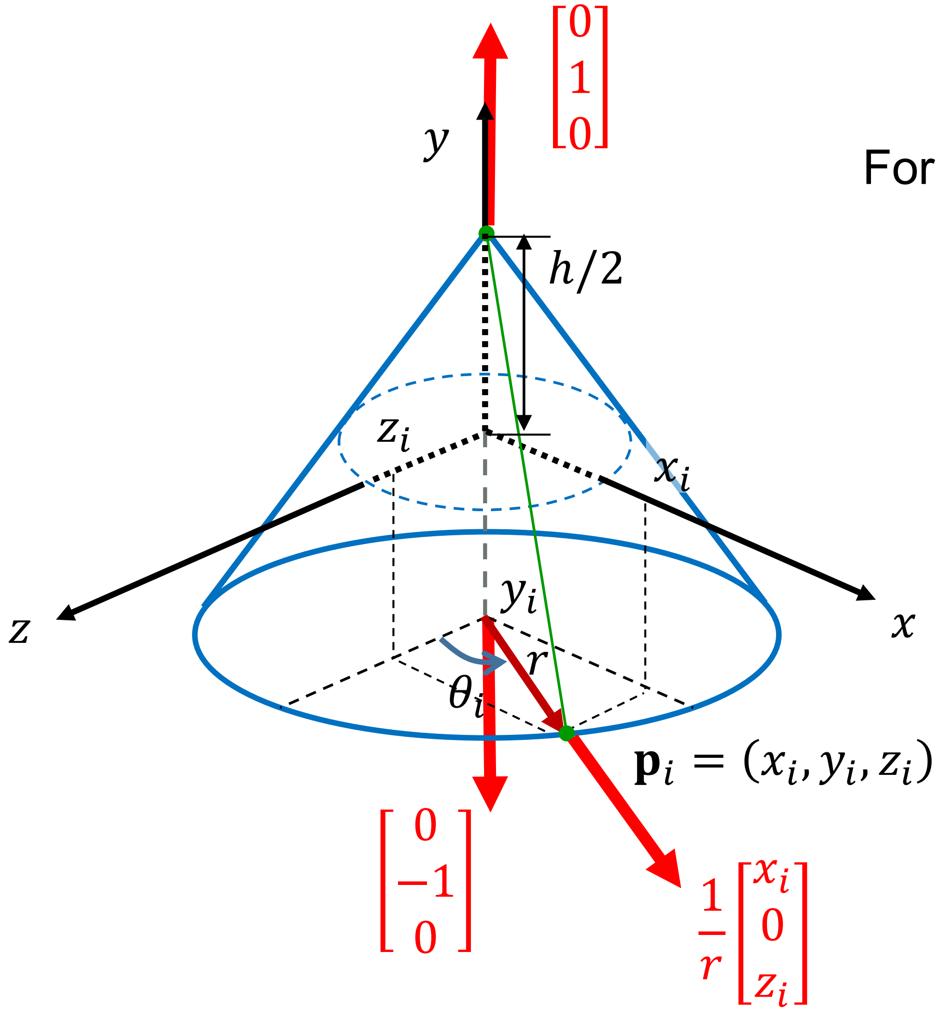
$$x_{i,j} = x_i^0 + \Delta x_{i,j} \qquad \theta_i = 2\pi \cdot i/n_0$$

$$y_j = r_1 \sin \phi_j \qquad \phi_j = 2\pi \cdot j/n_1$$

$$z_{i,j} = z_i^0 + \Delta z_{i,j}$$

$$\left[\begin{aligned} x_i^0 &= (r_0 + r_1)\sin\theta_i \\ z_i^0 &= (r_0 + r_1)\cos\theta_i \\ \\ \Delta x_{i,j} &= l_j \sin\theta_i \\ \Delta z_{i,j} &= l_j \cos\theta_i \\ \\ l_j &= r_1 \cos\phi_j \end{aligned}\right.$$

- **Cone Primitive**



$\begin{bmatrix}0\\1\\0\end{bmatrix}$
y
$h/2$
z_i
x_i
z
x
y_i
r
θ_i
$\mathbf{p}_i = (x_i, y_i, z_i)$
$\begin{bmatrix}0\\-1\\0\end{bmatrix}$
$\frac{1}{r}\begin{bmatrix}x_i\\0\\z_i\end{bmatrix}$

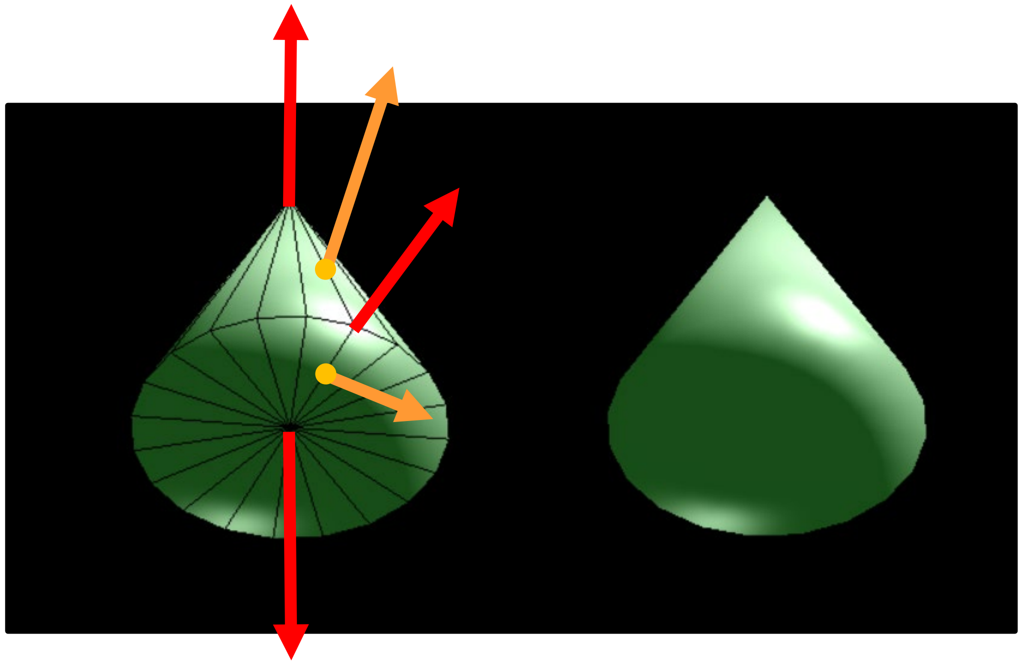
For $i = 0,1, \ldots, n$

$$\theta_i = 2\pi \cdot i/n$$

$$x_i = r \sin \theta_i$$

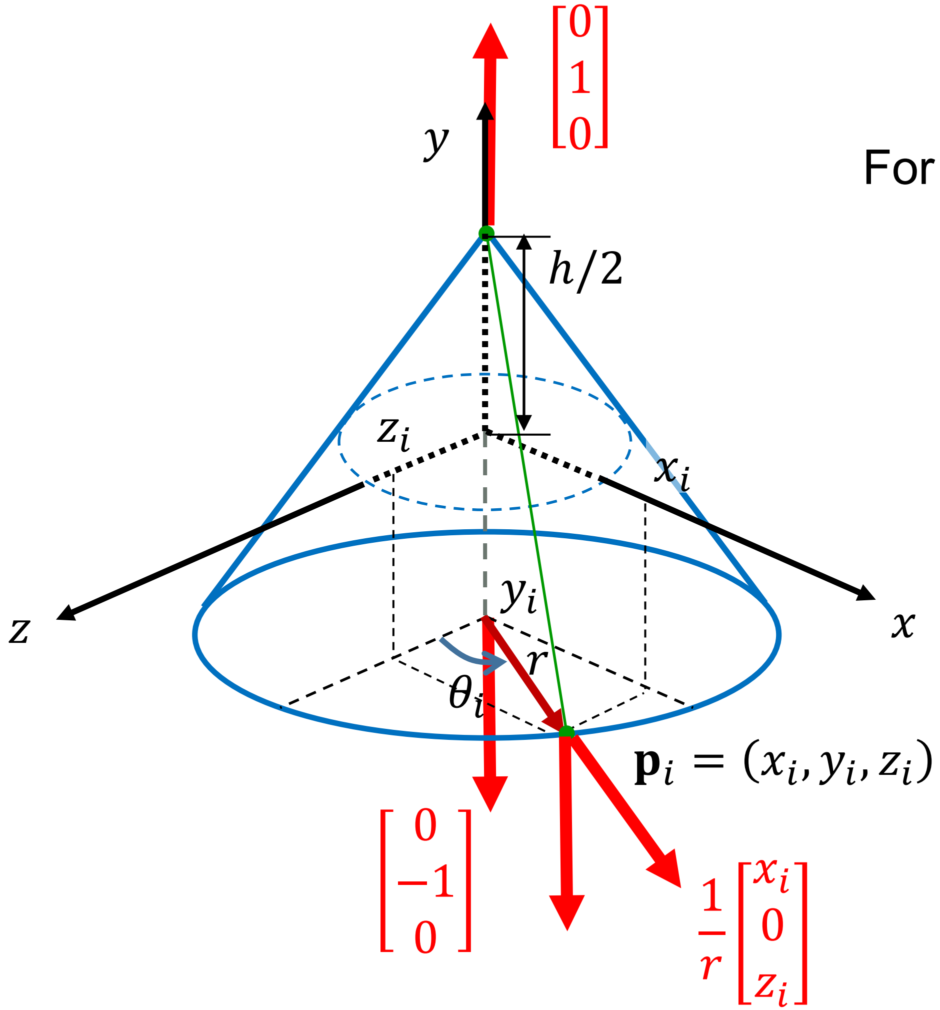
$$y_i = -h/2$$

$$z_i = r \cos \theta_i$$



Why? Any better way?

- **Cone Primitive**



$\begin{bmatrix}0\\1\\0\end{bmatrix}$
y
$h/2$
z_i
x_i
z
x
y_i
r
θ_i
$\mathbf{p}_i = (x_i, y_i, z_i)$
$\begin{bmatrix}0\\-1\\0\end{bmatrix}$
$\frac{1}{r}\begin{bmatrix}x_i\\0\\z_i\end{bmatrix}$

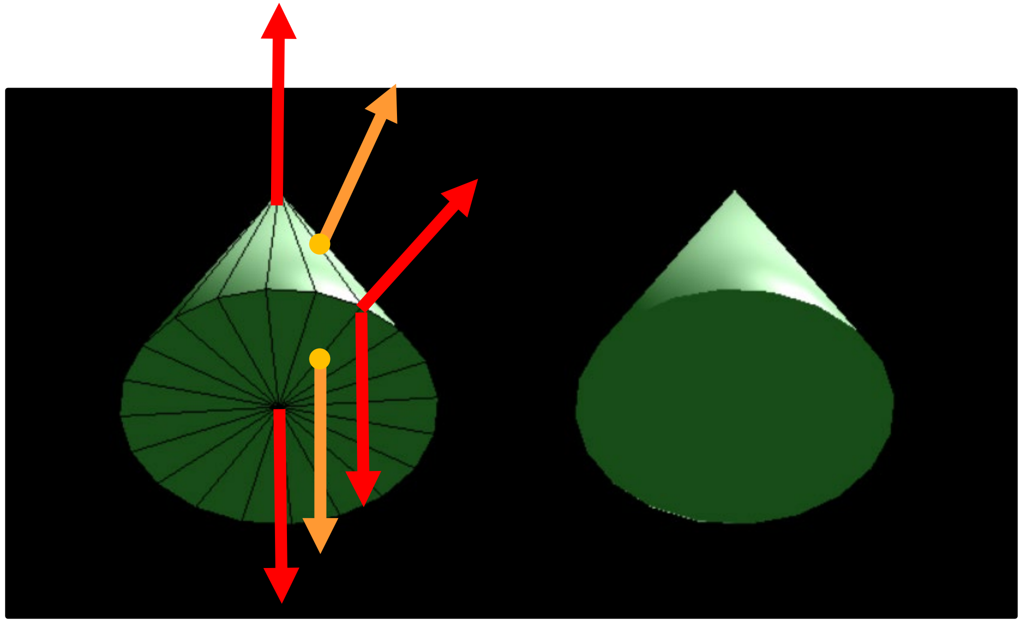
For $i = 0,1,\ldots,n$

$$\theta_i = 2\pi \cdot i/n$$

$$x_i = r\sin\theta_i$$

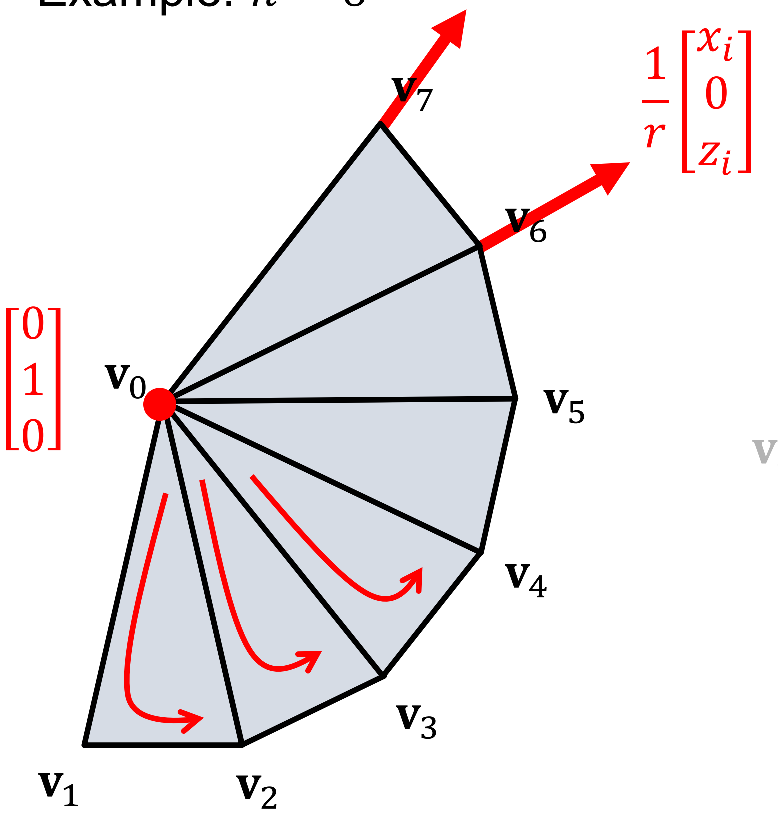
$$y_i = -h/2$$

$$z_i = r\cos\theta_i$$



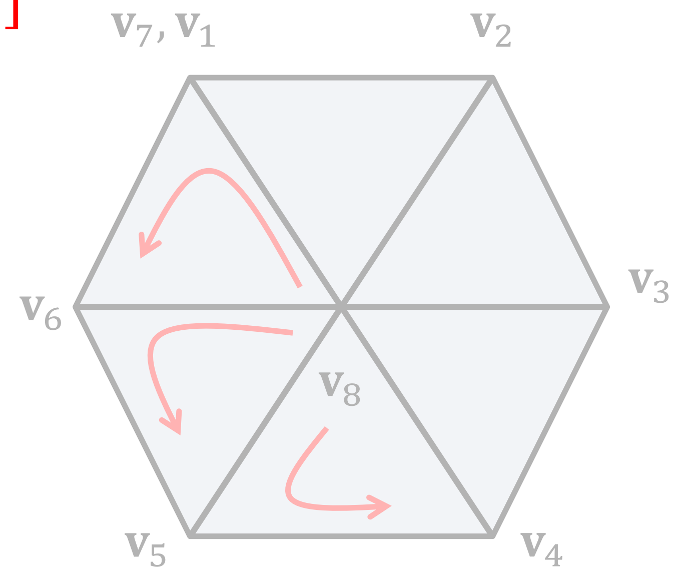
- Computing the sequence of vertices

* Example: $n = 6$



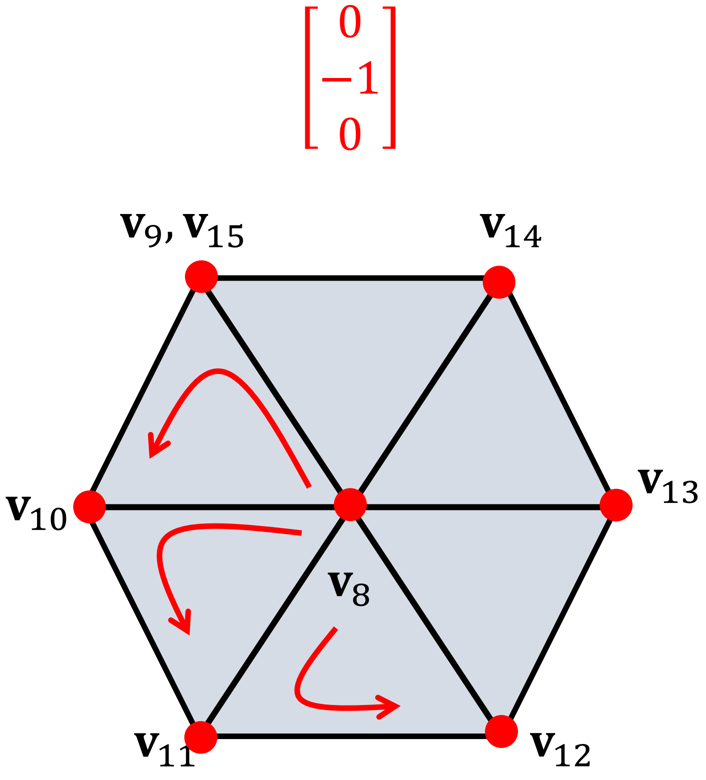
Side triangles represented by a ***triangle fan***

[0, 1, 2, 3, 4, …, n+1]



Bottom triangles represented by a ***triangle fan***

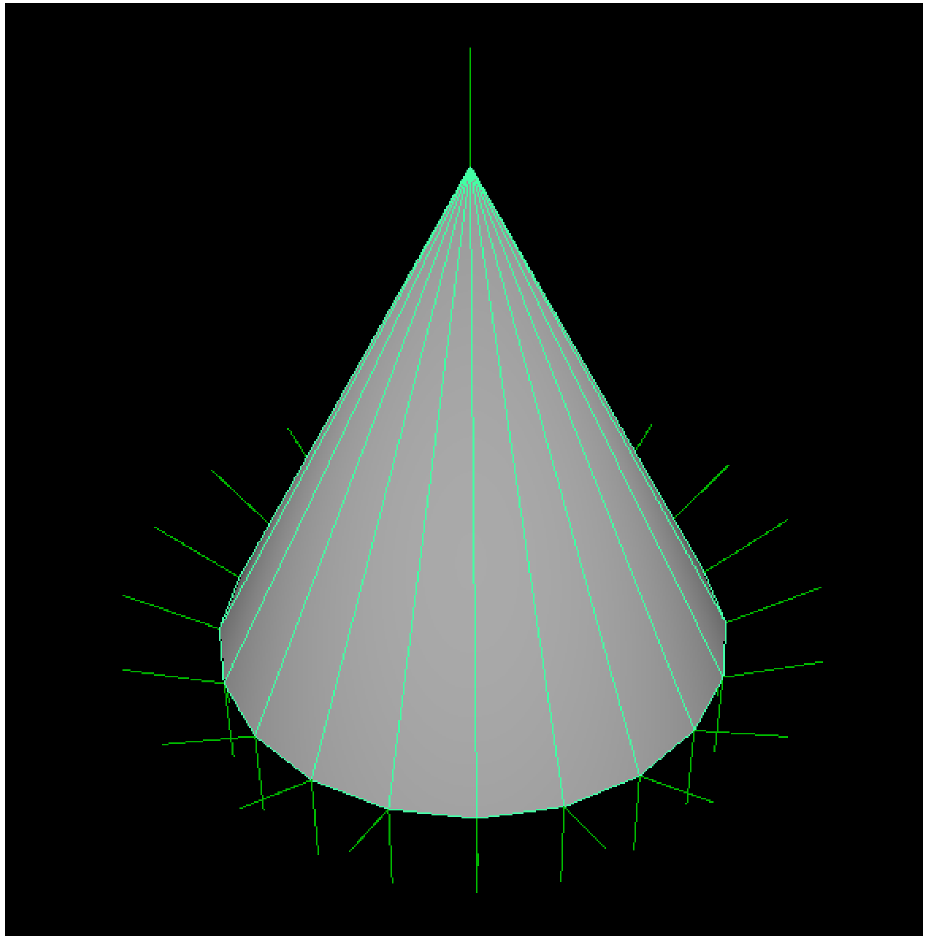
[n+2, n+1, …, 3, 2, 1]



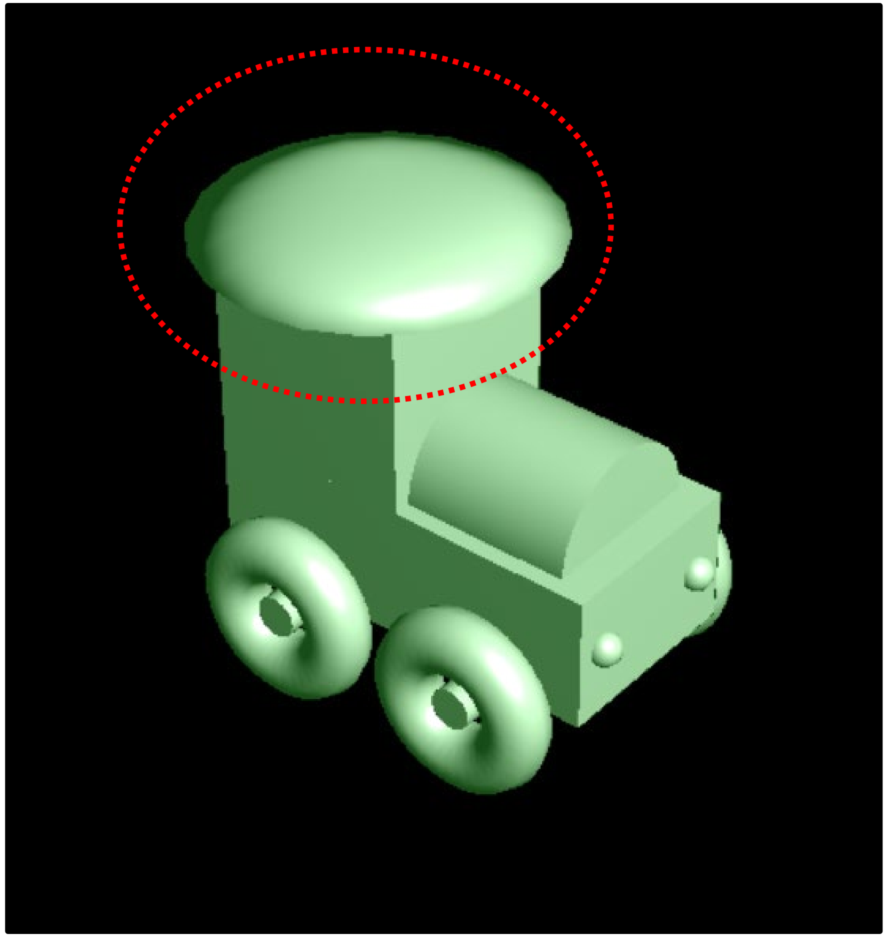
Bottom triangles represented by a ***triangle fan***

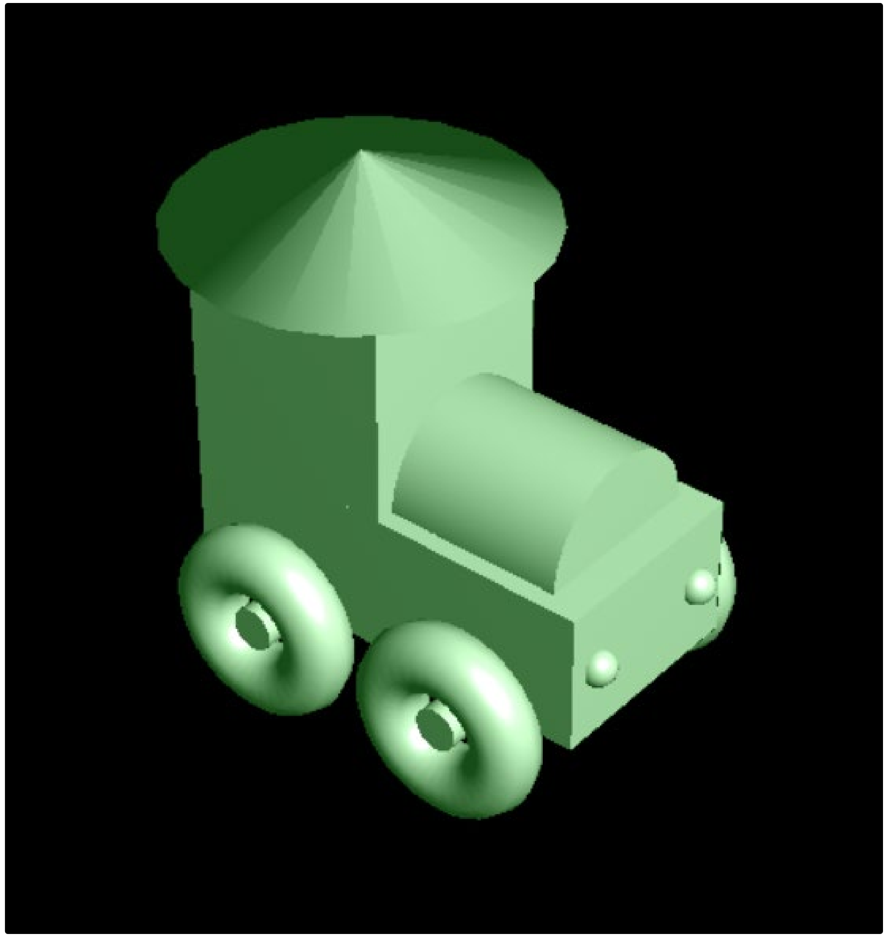
[n +2, n +3, …, $2n + 3$]

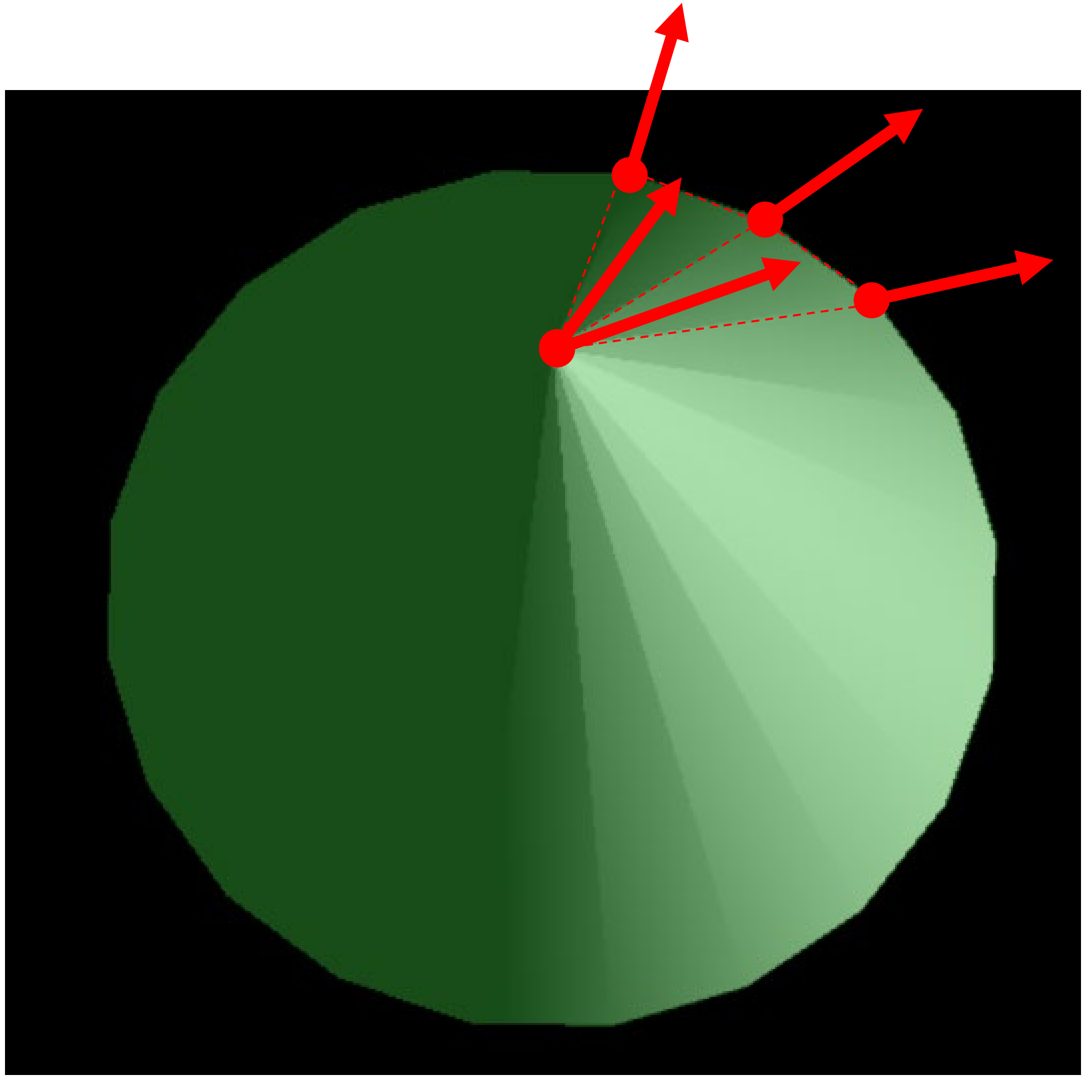
- Case Study: Cone Primitive in Autodesk Maya



- Is it perfect?

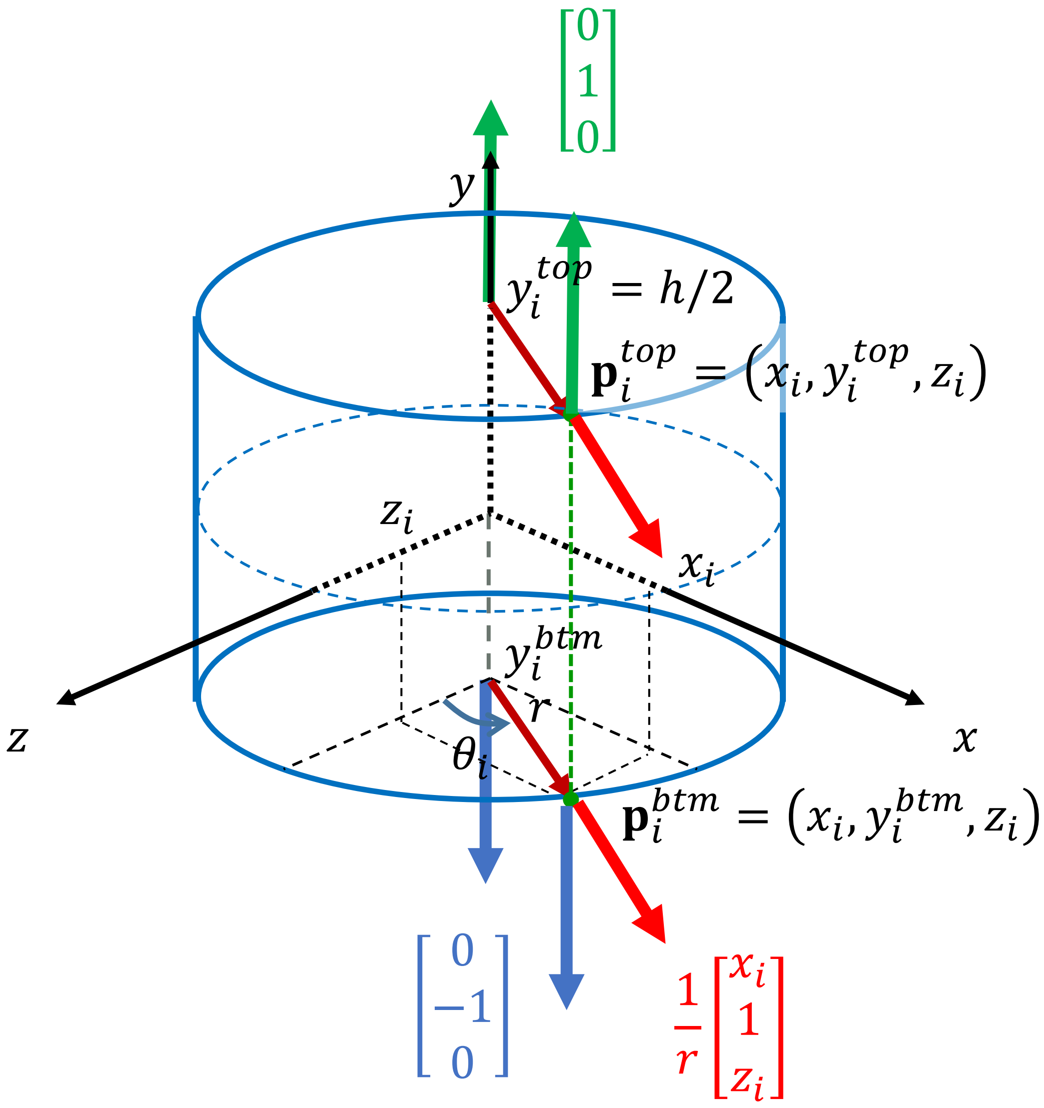






Some Mach band effect

- **Cylinder Primitive**



$\begin{bmatrix}0\\1\\0\end{bmatrix}$
y
$y_i^{top} = h/2$
$\mathbf{p}_i^{top} = (x_i, y_i^{top}, z_i)$
z_i
x_i
y_i^{btm}
r
θ_i
z
x
$\mathbf{p}_i^{btm} = (x_i, y_i^{btm}, z_i)$
$\begin{bmatrix}0\\-1\\0\end{bmatrix}$
$\frac{1}{r}\begin{bmatrix}x_i\\1\\z_i\end{bmatrix}$

For $i = 0,1,\ldots,n$

$$\theta_i = 2\pi \cdot i/n$$

$$x_i = r\sin\theta_i$$

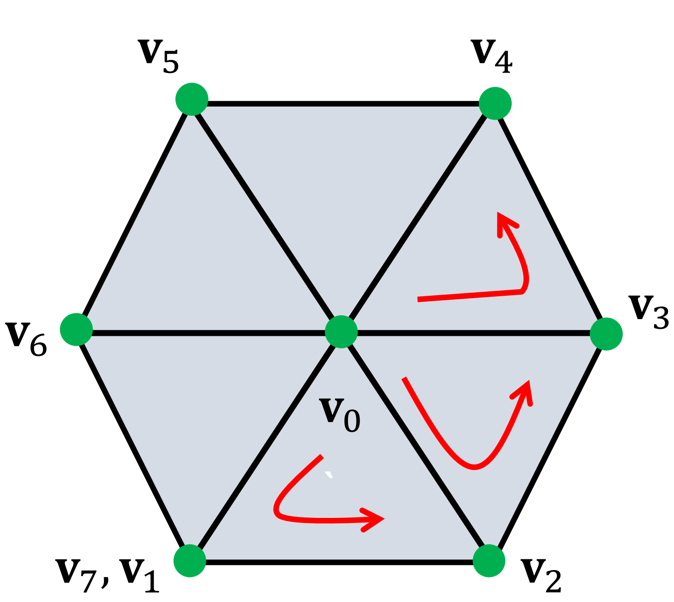
$$y_i^{top} = h/2$$

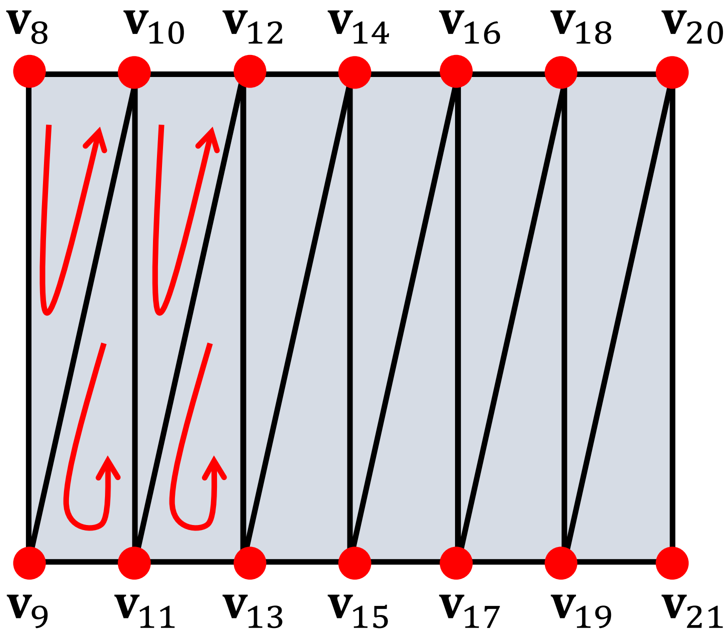
$$y_i^{btm} = -h/2$$

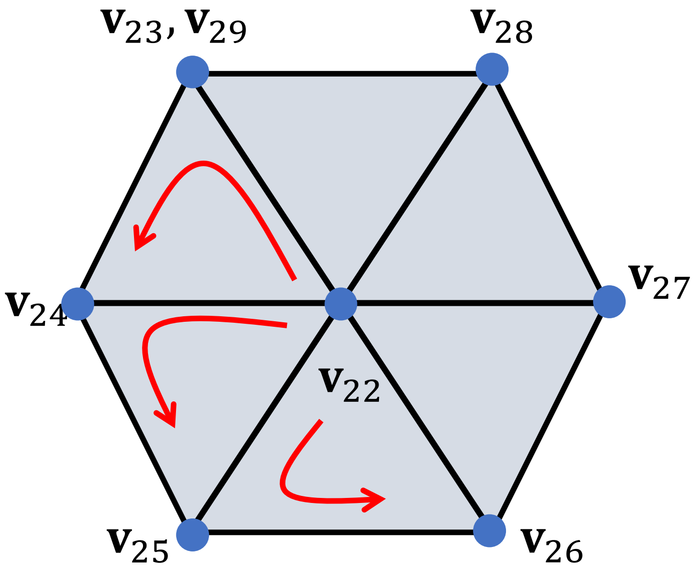
$$z_i = r\cos\theta_i$$

- Computing the sequence of vertices

* Example: $n = 6$







Top triangles represented by a ***triangle fan***

$[0, 1, 2, \ldots, n + 1]$

Side triangles represented by a ***triangle strip***

$[n + 2, n + 3\ , \ldots, 3n + 3]$

Bottom triangles represented by a ***triangle fan***

$[3n + 4, 4n + 5, \ldots, 4n + 5]$

Gouraud Shading

- Vertex shader

```
......

in vec4 vPosition;
in vec4 vNormal;
in vec4 vColor;
out vec4 fColor;

vec3 Ia = vec3(0.3, 1.0, 0.3);
vec3 Il = vec3(1.0, 1.0, 1.0);
float Ka = 0.3;
float Ks = 0.5;
float Kd = 0.8;
float c[3] = {0.01, 0.001, 0.0};
float n = 10.0;
vec4 LightPos_wc = vec4(10, 10, 3, 1);
```

The normal data for the current primitive should be given through variable **vNormal**.

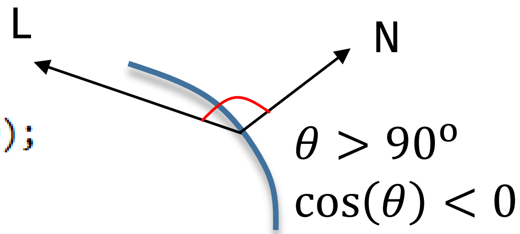
$$I = I_a k_a + f_{att} I_l (k_d \cos\theta + k_s \cos^n \alpha)$$

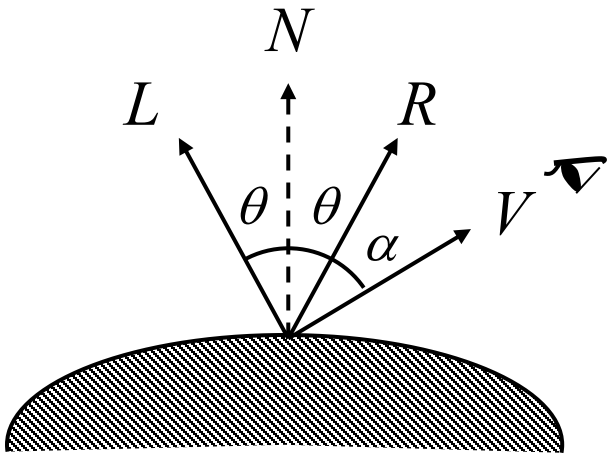
Coefficients for attenuation factor f_{att}

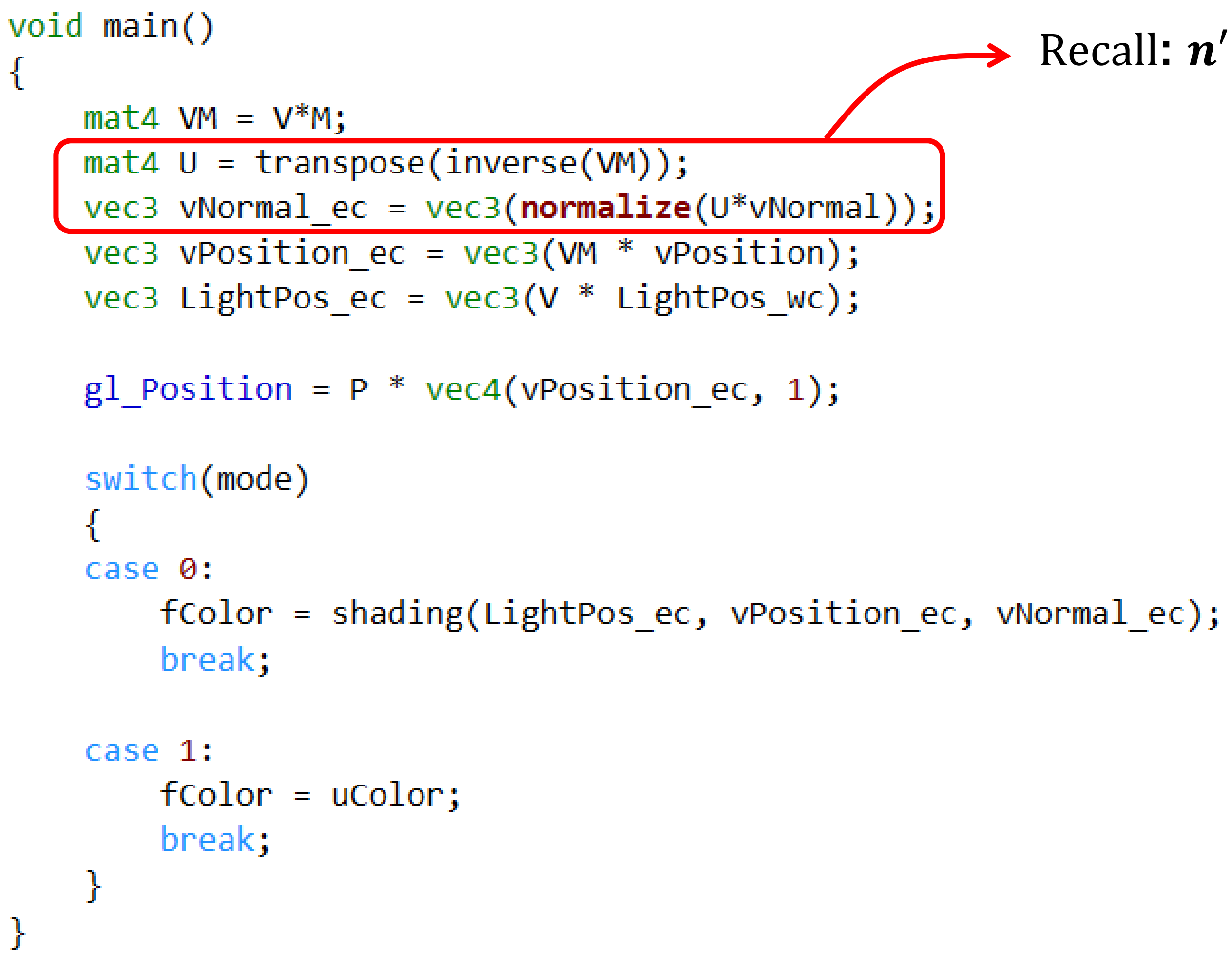
Light position with respect to the world coordinate system.

(continued in the next slides…)

```
vec4 shading(vec3 LightPos_ec, vec3 vPosition_ec, vec3 vNormal_ec)
{
    vec3 N = normalize(vNormal_ec);
    vec3 L = LightPos_ec - vPosition_ec;
    float d = length(L); L = L/d;
    vec3 V = normalize(vec3(0.0) - vPosition_ec);
    vec3 R = reflect(-L, N);

    float fatt = min(1.0 / (c[0] + c[1]*d + c[2]*d*d), 1.0);

    float cos_theta = max(dot(N,L),0);
    float cos_alpha = max(dot(V,R),0);

    vec3 I = Ia * Ka + fatt * Il * (Kd * cos_theta + Ks * pow(cos_alpha, n));

    return vec4(I,1);
}
```



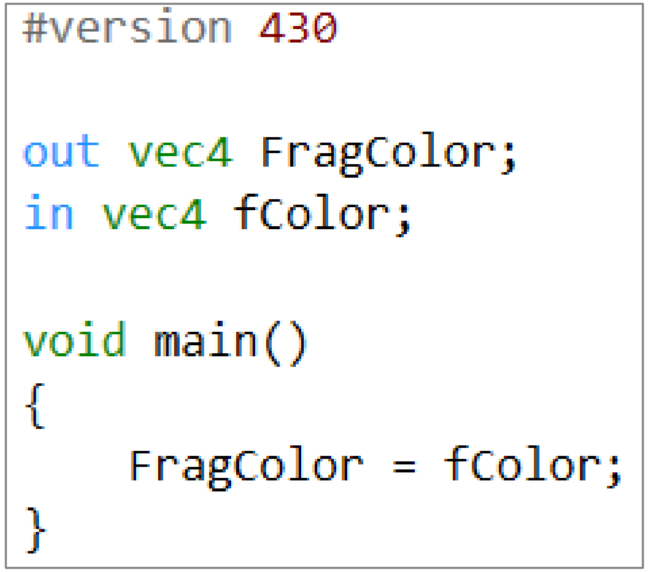




```
void main()
{
    mat4 VM = V*M;
    mat4 U = transpose(inverse(VM));
    vec3 vNormal_ec = vec3(normalize(U*vNormal));
    vec3 vPosition_ec = vec3(VM * vPosition);
    vec3 LightPos_ec = vec3(V * LightPos_wc);

    gl_Position = P * vec4(vPosition_ec, 1);

    switch(mode)
    {
    case 0:
        fColor = shading(LightPos_ec, vPosition_ec, vNormal_ec);
        break;

    case 1:
        fColor = uColor;
        break;
    }
}
```

Recall: $\boldsymbol{n}' = (\boldsymbol{M}^{-1})^{\boldsymbol{T}} \cdot \boldsymbol{n}$

- Fragment shader

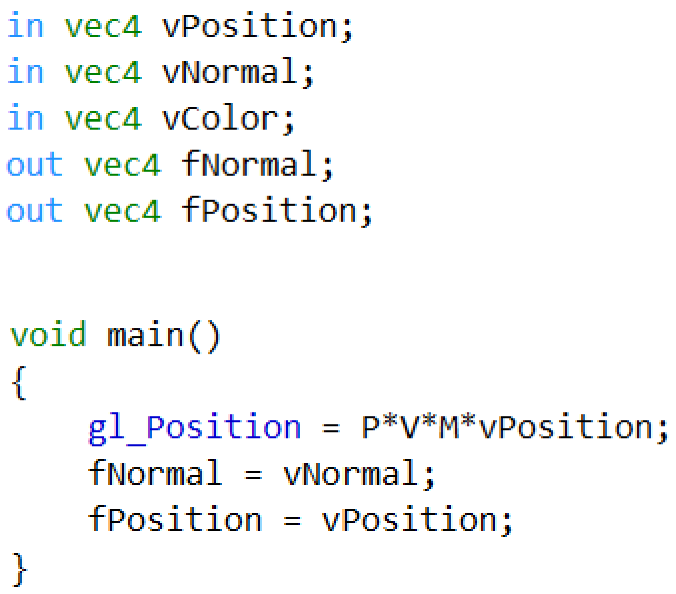


```
#version 430

out vec4 FragColor;
in vec4 fColor;

void main()
{
    FragColor = fColor;
}
```

Phong Shading

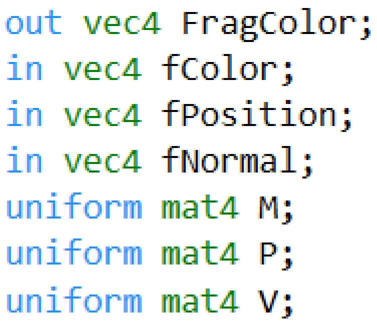
- Vertex shader

……

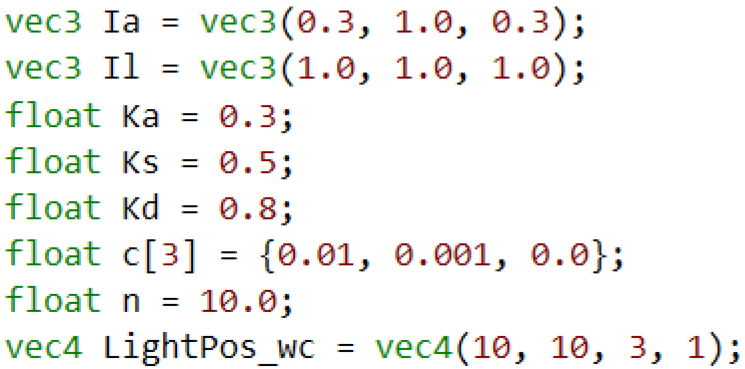


```
in vec4 vPosition;
in vec4 vNormal;
in vec4 vColor;
out vec4 fNormal;
out vec4 fPosition;


void main()
{
    gl_Position = P*V*M*vPosition;
    fNormal = vNormal;
    fPosition = vPosition;
}
```

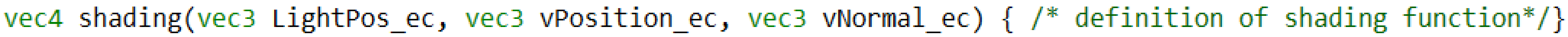
- Fragment shader



```
out vec4 FragColor;
in vec4 fColor;
in vec4 fPosition;
in vec4 fNormal;
uniform mat4 M;
uniform mat4 P;
uniform mat4 V;
```



```
vec3 Ia = vec3(0.3, 1.0, 0.3);
vec3 Il = vec3(1.0, 1.0, 1.0);
float Ka = 0.3;
float Ks = 0.5;
float Kd = 0.8;
float c[3] = {0.01, 0.001, 0.0};
float n = 10.0;
vec4 LightPos_wc = vec4(10, 10, 3, 1);
```



```
vec4 shading(vec3 LightPos_ec, vec3 vPosition_ec, vec3 vNormal_ec) { /* definition of shading function*/}
```

(continued in the next slides…)

```
void main()
{
    mat4 VM = V*M;
    mat4 U = transpose(inverse(VM));
    vec3 vNormal_ec = vec3(normalize(U*fNormal));
    vec3 fPosition_ec = vec3(VM * fPosition);
    vec3 LightPos_ec = vec3(V * LightPos_wc);

    FragColor = shading(LightPos_ec, fPosition_ec, vNormal_ec);
}
```

- What to submit:
  - A **zip file** that compresses the following files:
    - **Project source files** except libraries.
      - Clean your project before compression by selecting **Build → Clean Solution** in the main menu.
  - Please add comments to your GLSL code to show your understanding.
  - File name format
    - **hw10_000000.zip**, where 000000 must be replaced by your own student ID.
- Due date: **To be announced later**