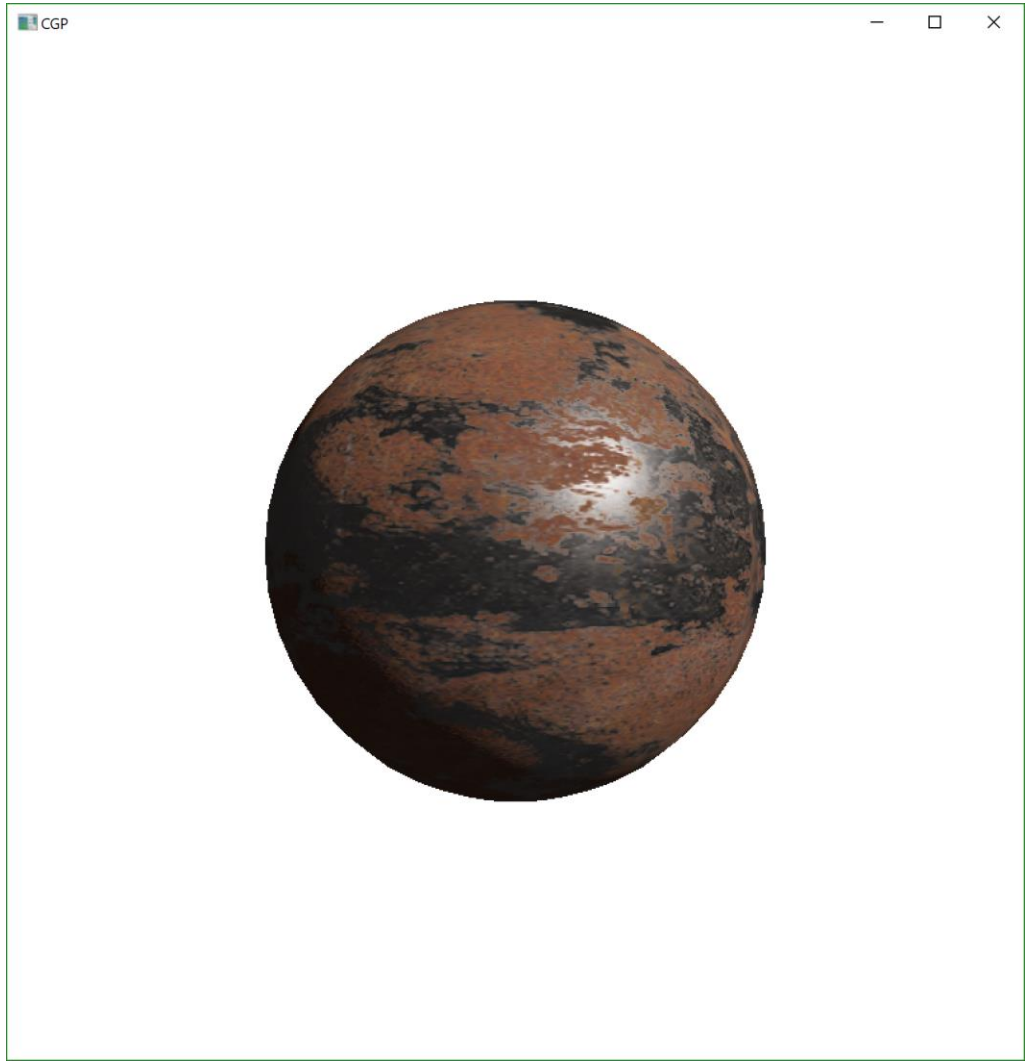
# CGP 3

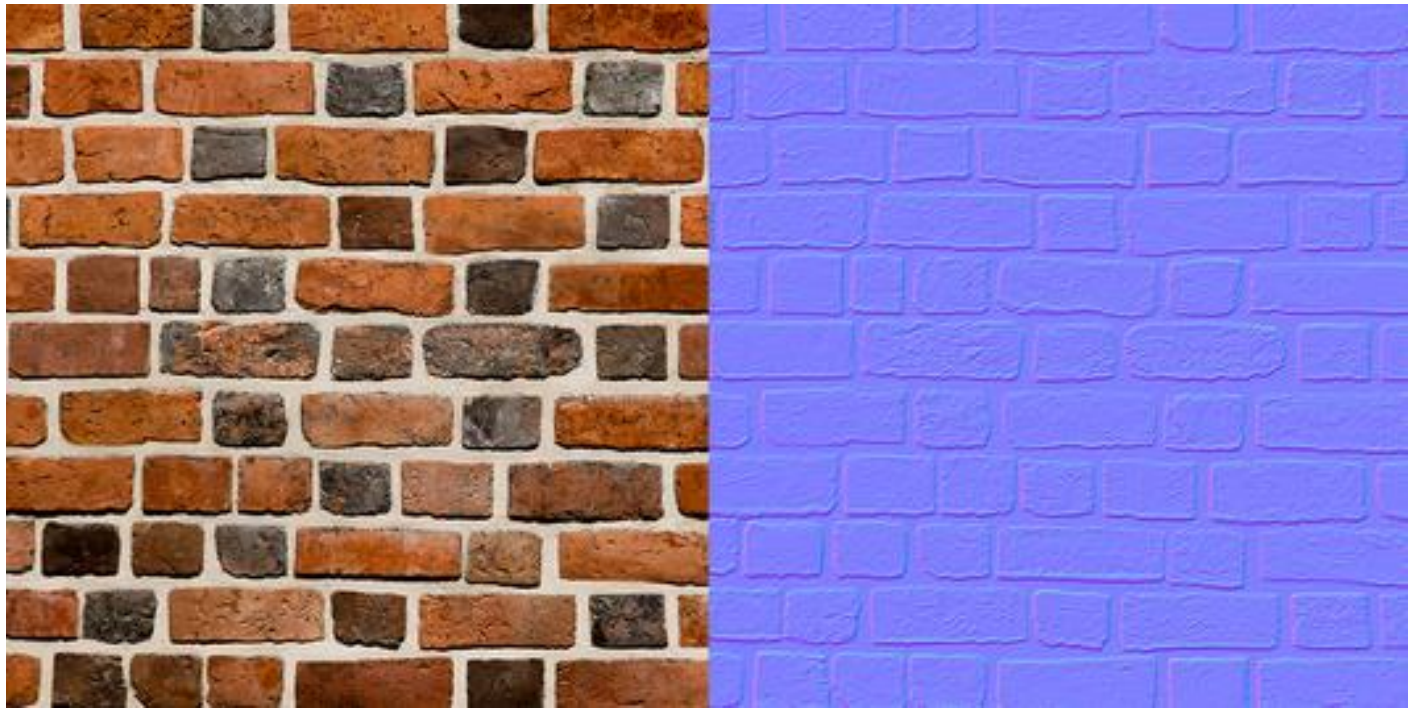
Dr W. Palubicki

# Textures and microfacet BRDF

- Download BRDF texture package and apply roughness and metallic parameter values stored in texel data



# Normal mapping

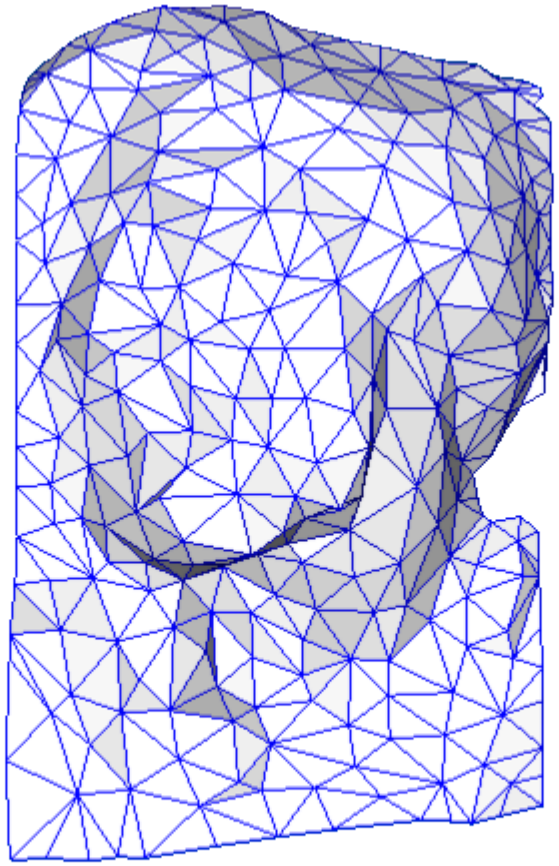


Texture

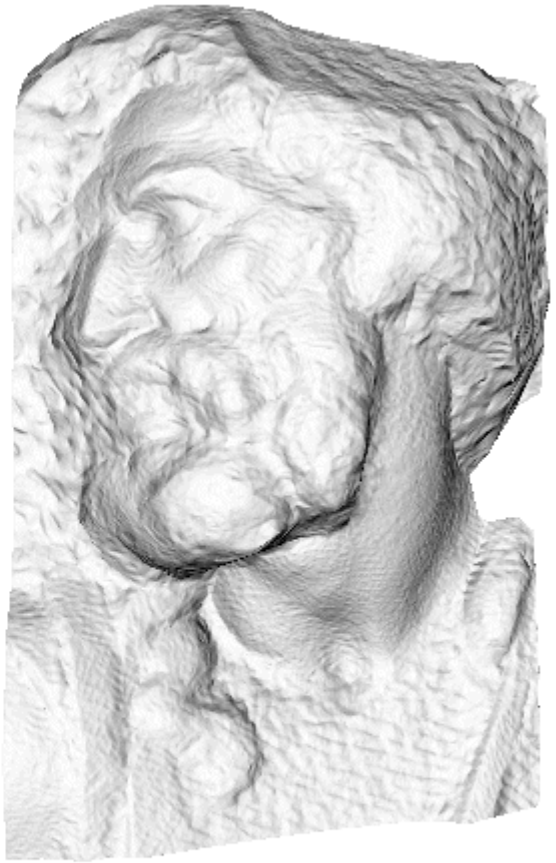
Normal map

# Normal mapping

original mesh
4M triangles

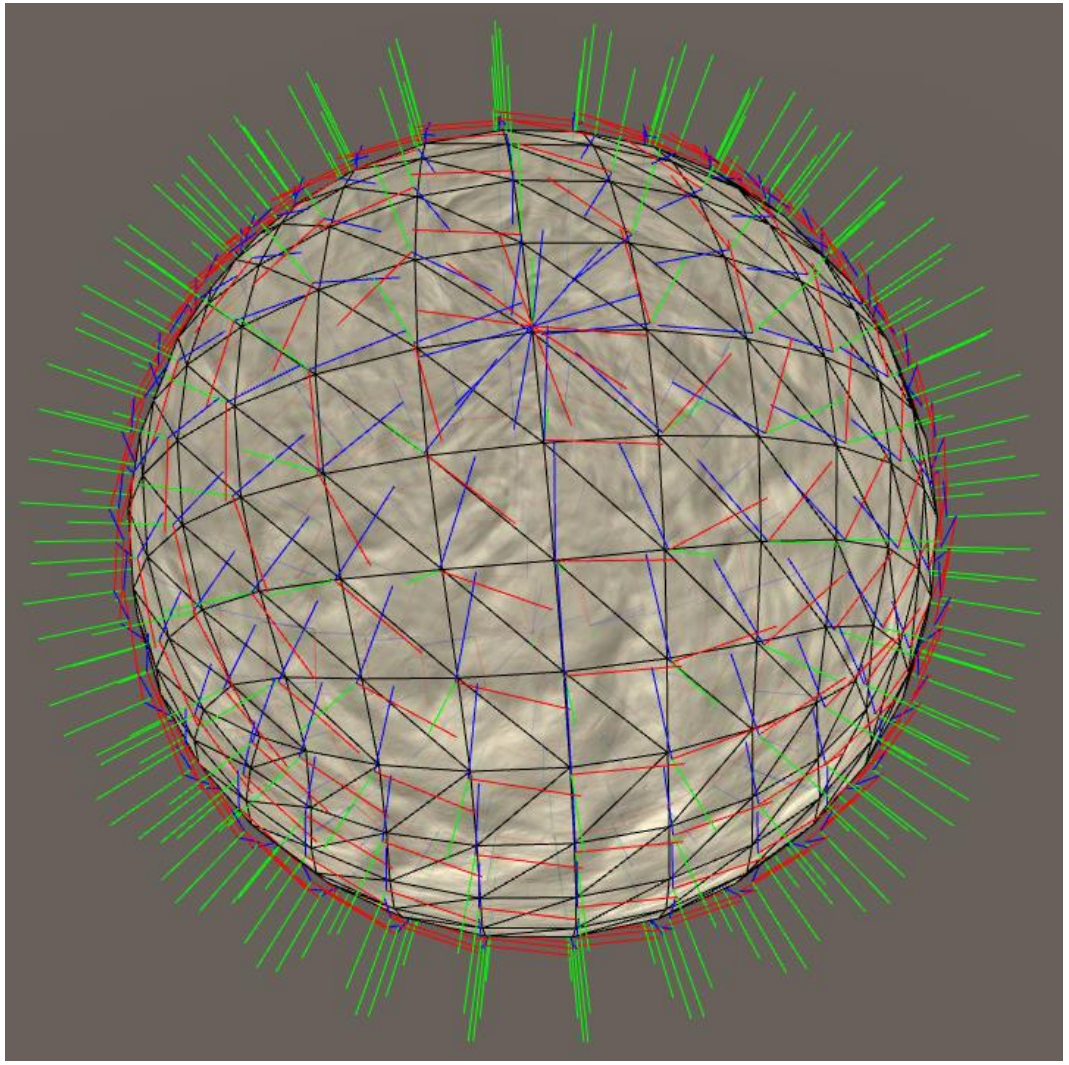


simplified mesh
500 triangles



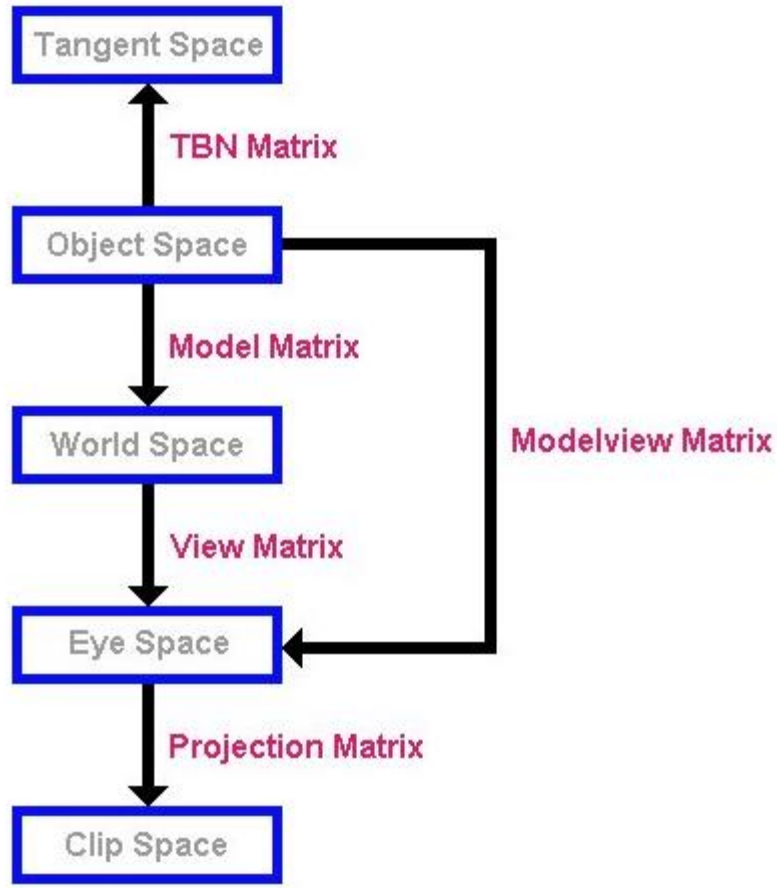
simplified mesh
and normal mapping
500 triangles

# Tangent space



# Calculating tangents in the case of a sphere

- **unit vector** x **surface normal** (choose nonidentical unit vector)
- Send tangent information to vertex shader


Tangent Space
TBN Matrix
Object Space
Model Matrix
World Space
Modelview Matrix
View Matrix
Eye Space
Projection Matrix
Clip Space

# OpenGL

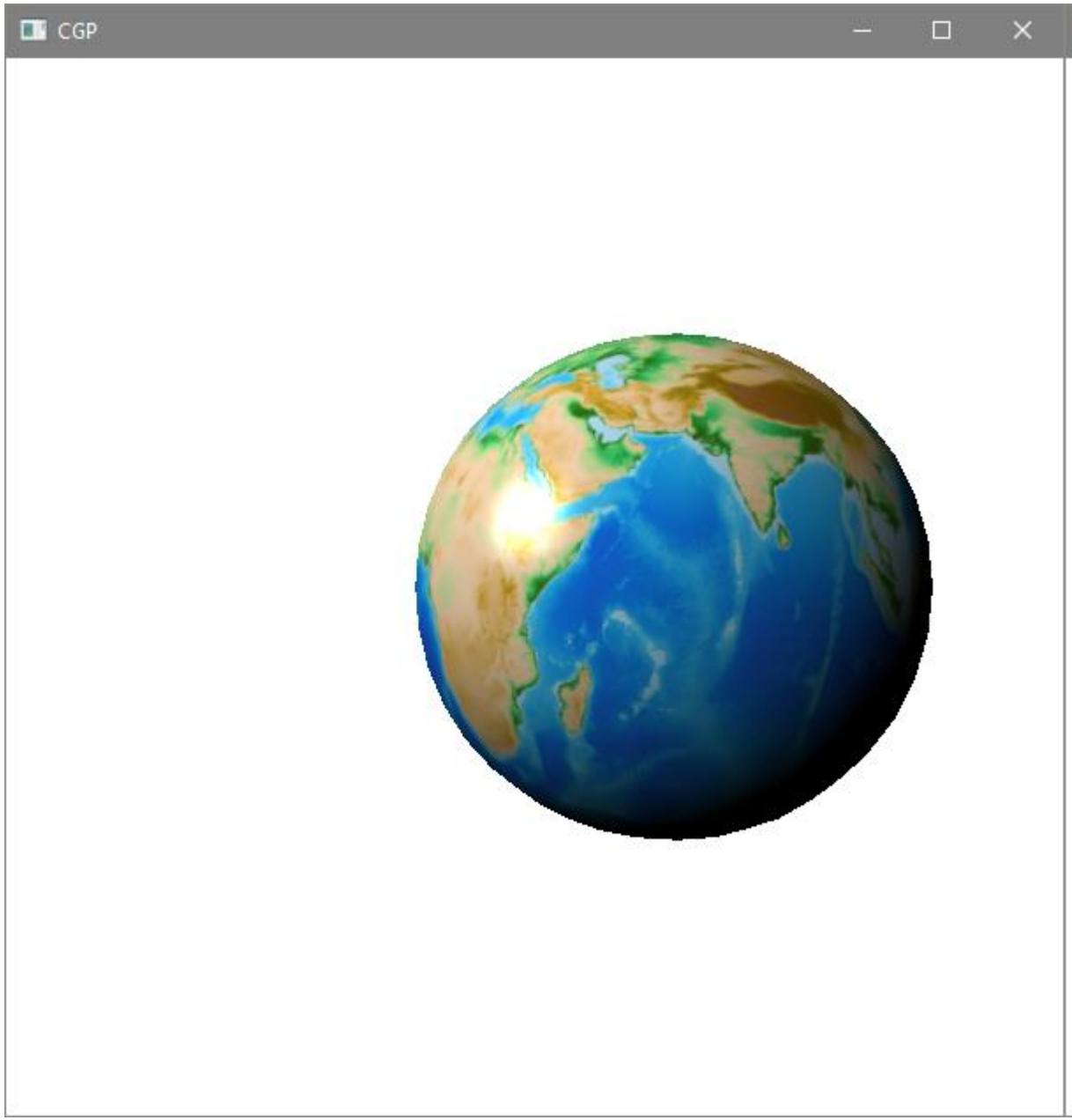
- A new function drawObjectTextureNormal has been added that takes a normal map as input (in addition to a texture).
- SetActiveTexture is used to specify the normal map in the shader sampler: `Core::SetActiveTexture(normalMap, "normalMap", program, 1);`
- Look up what the DrawModelT function does and how it differs to DrawModel.

# Vertex Shader

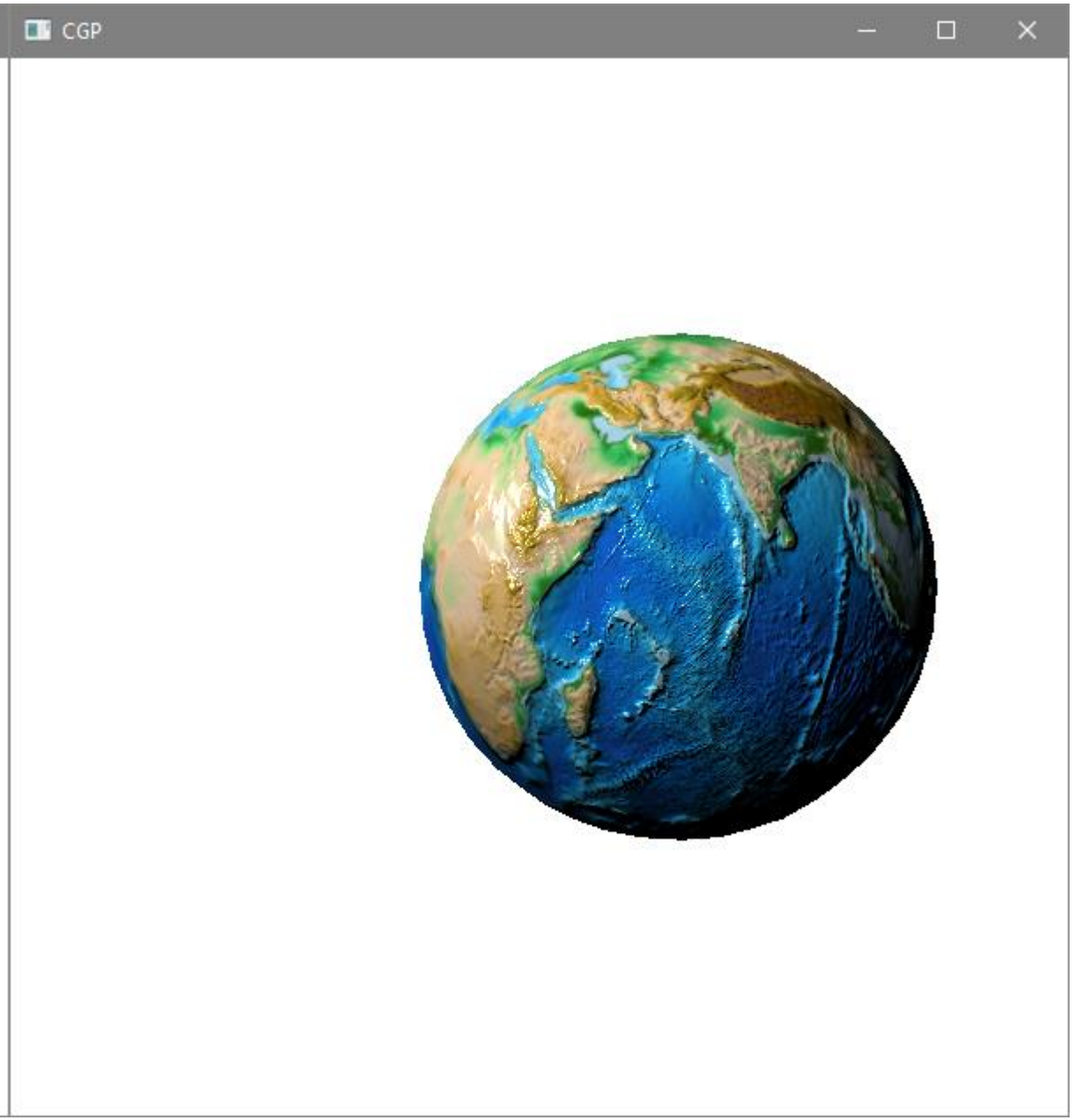
- Create a vertex shader with 4 attributes
  - layout(location = 0) in vec3 vertexPosition;
  - layout(location = 1) in vec2 vertexTexCoord;
  - layout(location = 2) in vec3 vertexNormal;
  - layout(location = 3) in vec3 vertexTangent;
- Calculate the normal, tangent and bitangent in world space (multiply modelMatrix with normal and tangent vectors – bitangent is the cross of transformed normal and tangent)
- Transform light, camera position and vertex position by tangent basis, e.g.
  - l.x = dot (lightDir, t);
  - l.y = dot (lightDir, b);
  - l.z = dot (lightDir, n);
- Pass the transformed vectors to fragment shader

# Fragment Shader

- Create 2 sampler2D variables for texture and normal map
- Instead of the interpolated normal use the normal stored in the normal map (you have to scale the normal $[0,1]^3 \rightarrow [-1,1]^3$)

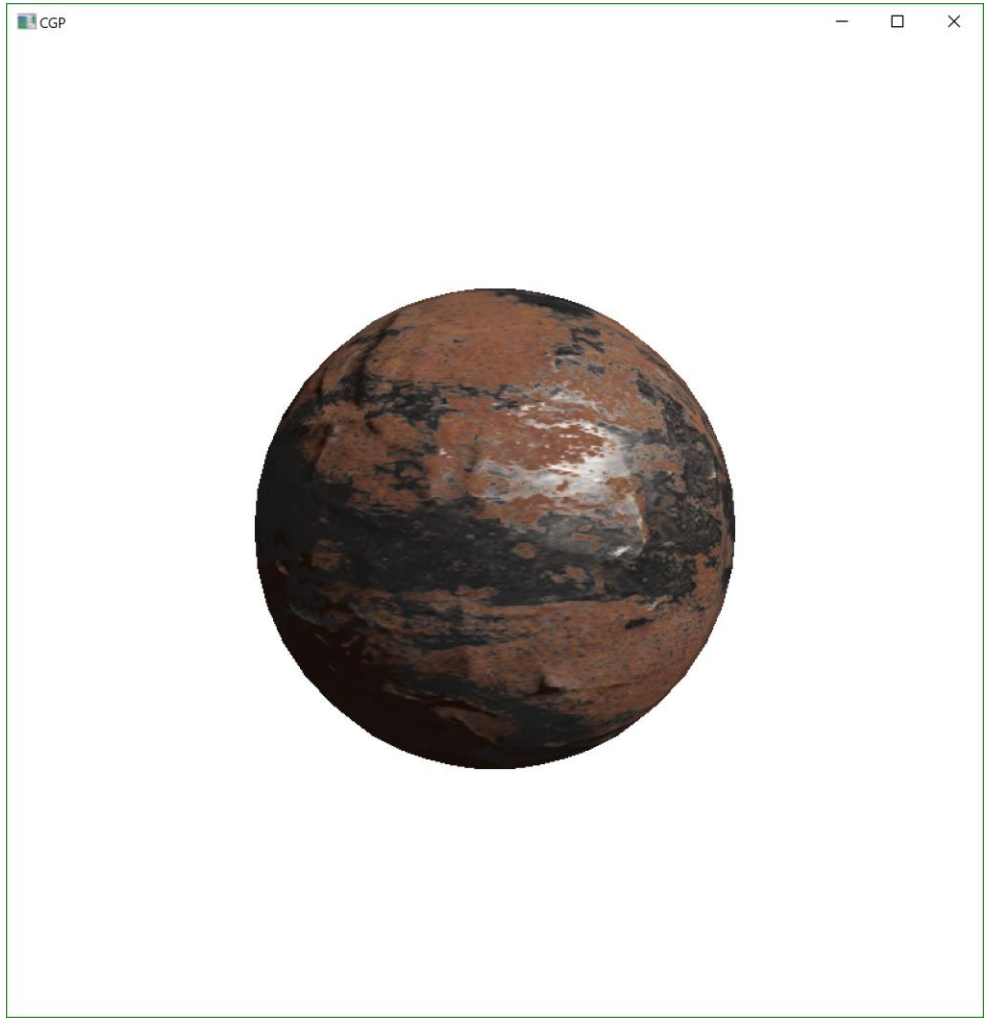



Without normal map



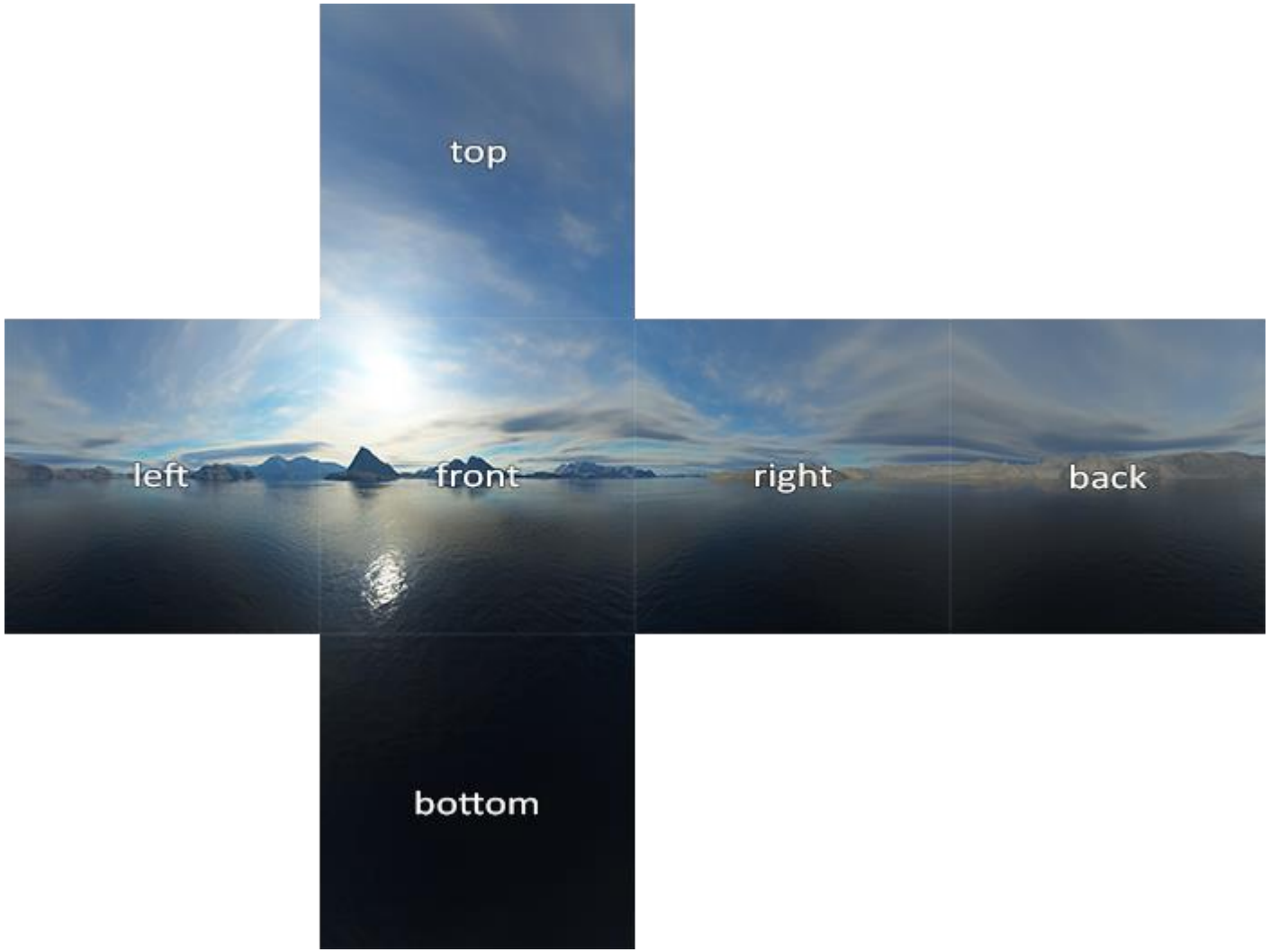

With normal map

# Normal-mapped sphere with microfacet textures

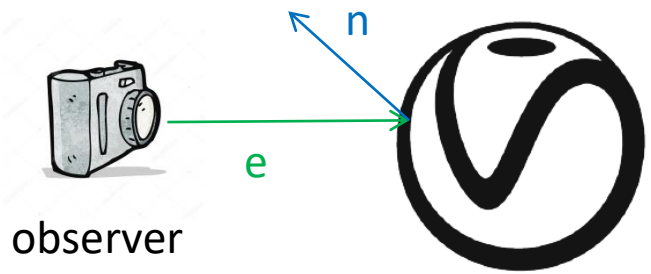


# Environment mapping

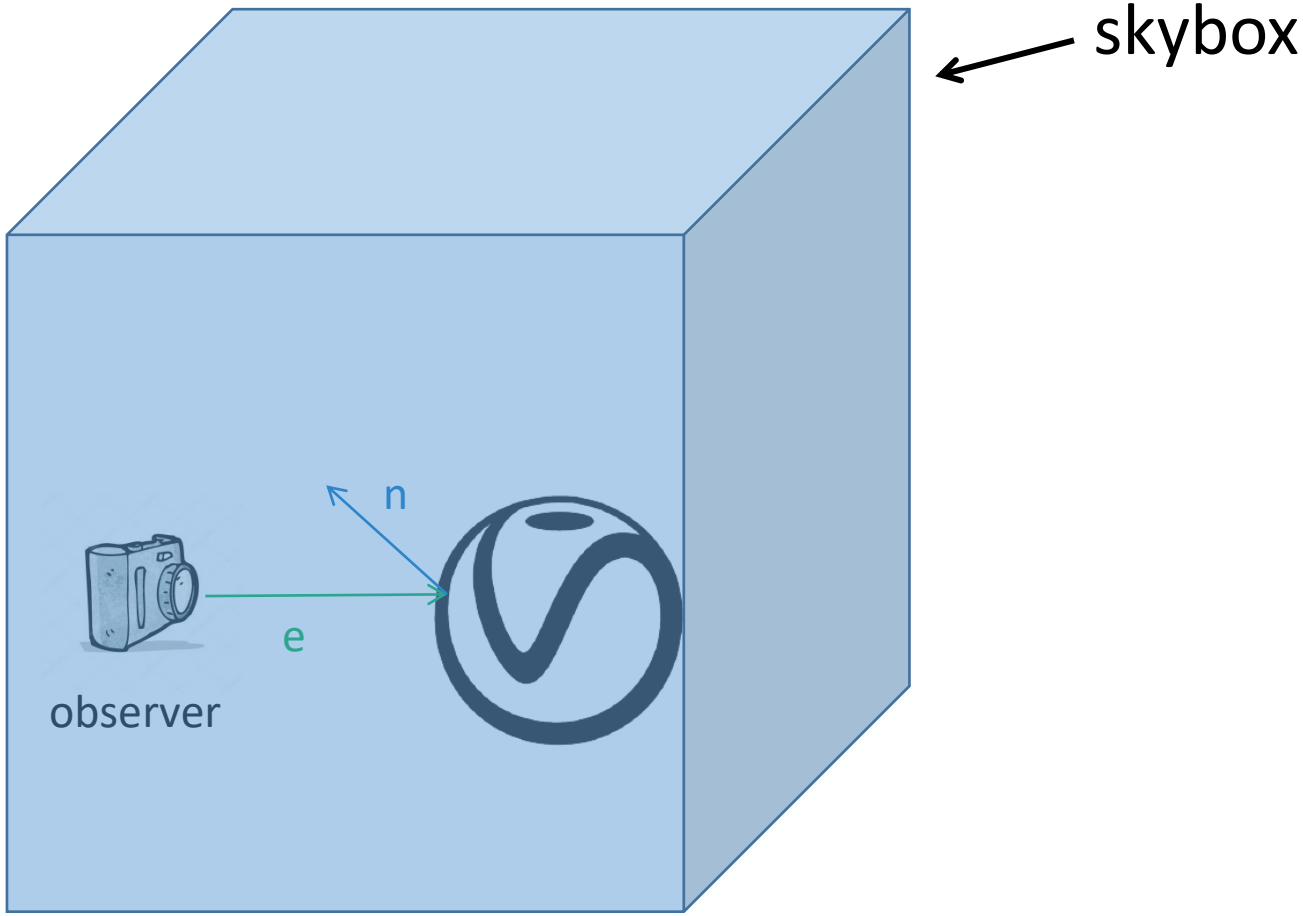
# Skybox



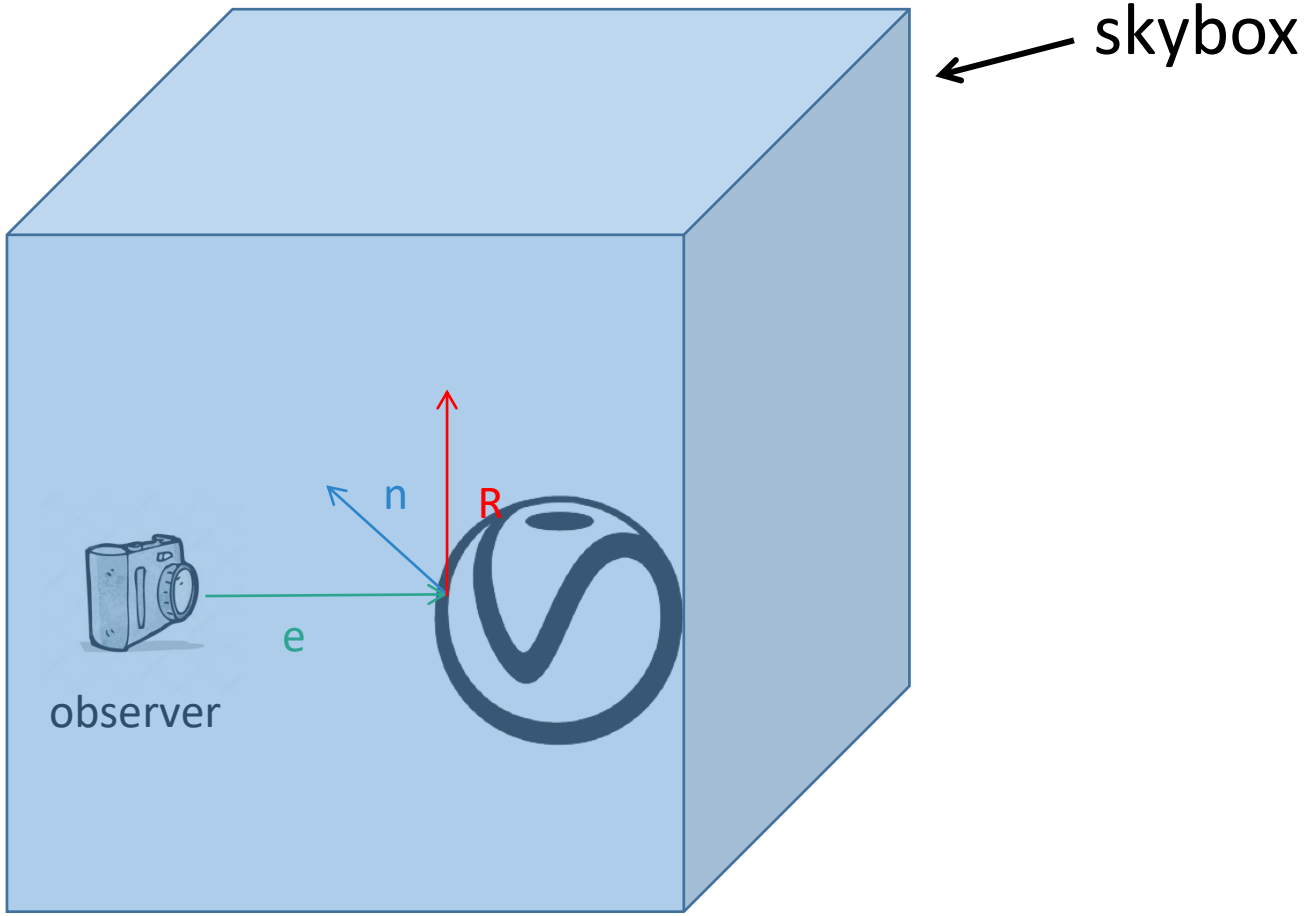
# Idea: environment mapping



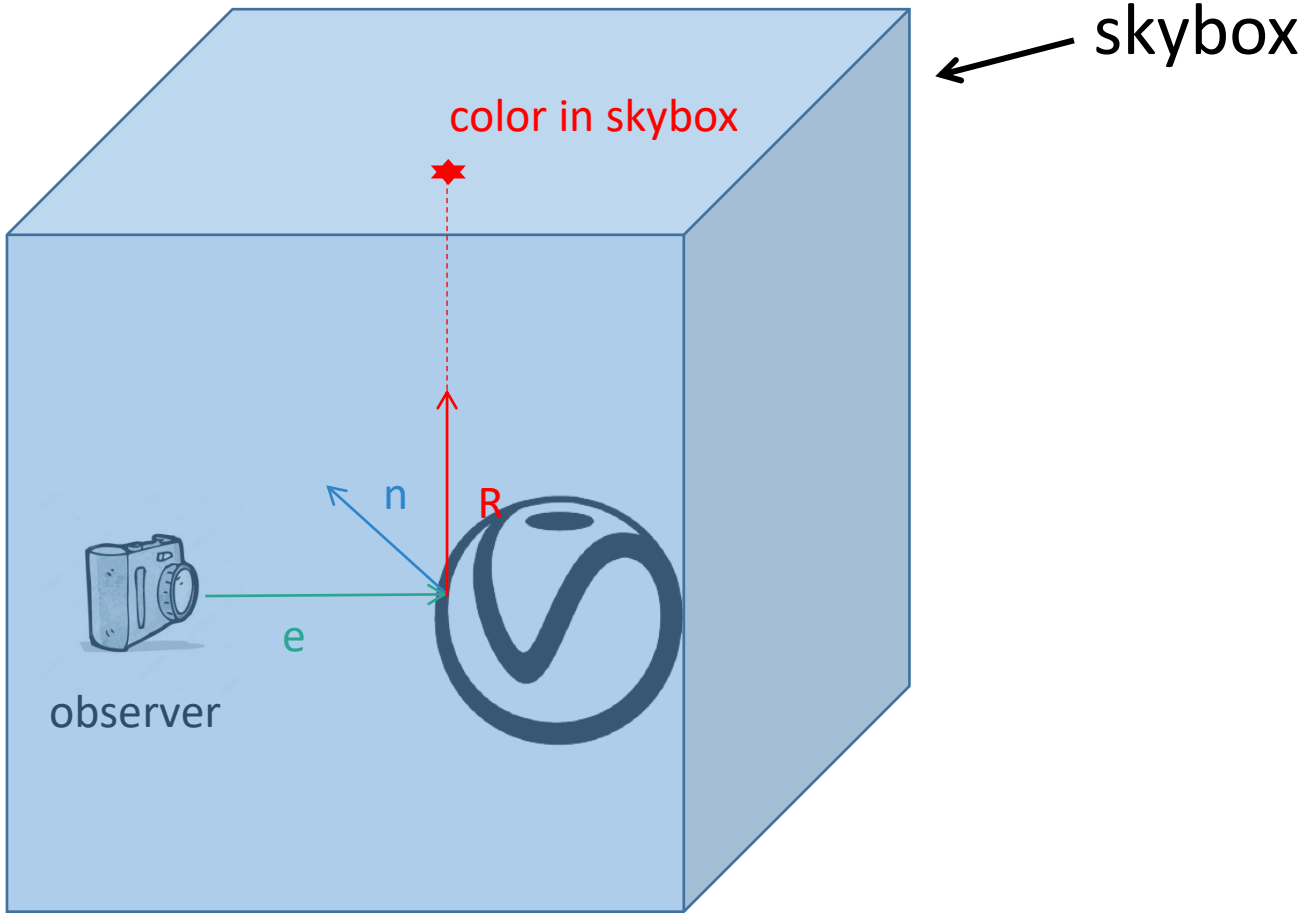
# Idea: environment mapping



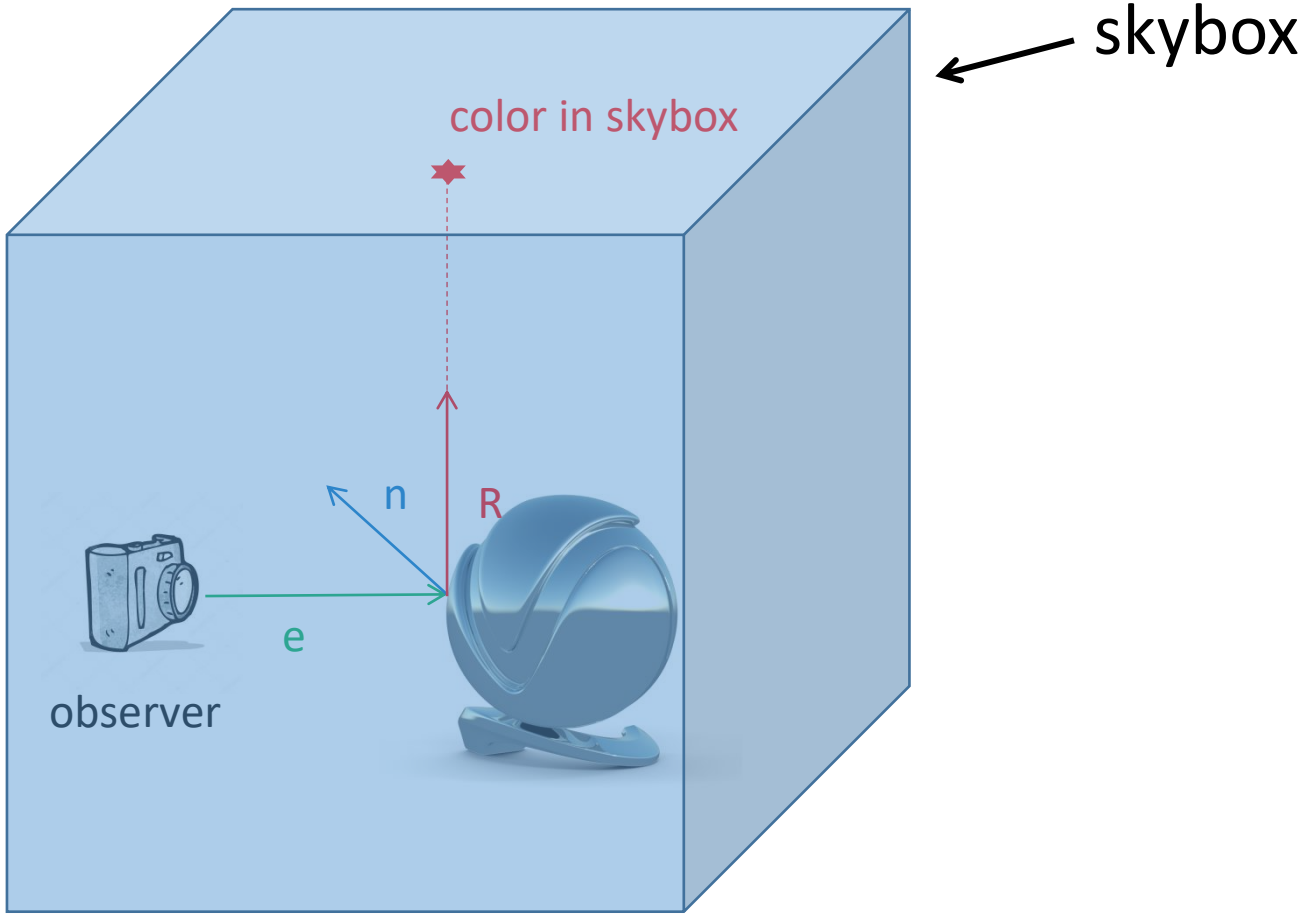
# Idea: environment mapping



# Idea: environment mapping

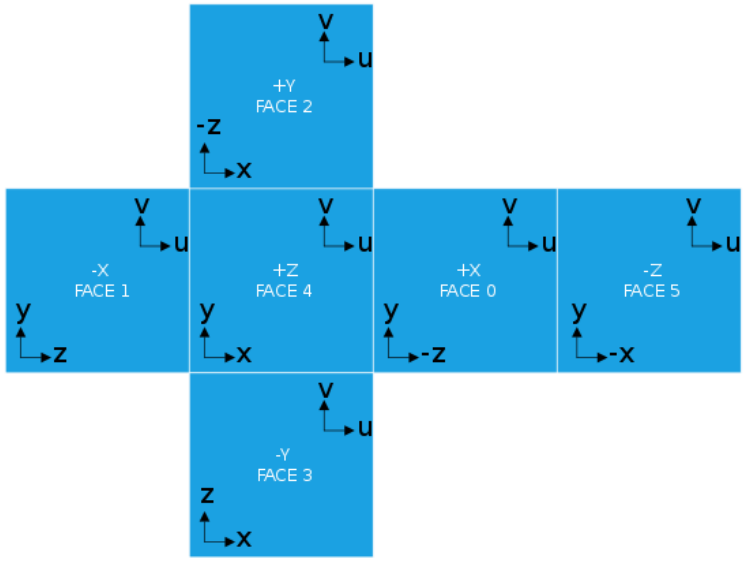


# Idea: environment mapping



# 6 Textures define cube map

GL_TEXTURE_CUBE_MAP_POSITIVE_X,
GL_TEXTURE_CUBE_MAP_NEGATIVE_X,
GL_TEXTURE_CUBE_MAP_POSITIVE_Y,
GL_TEXTURE_CUBE_MAP_NEGATIVE_Y,
GL_TEXTURE_CUBE_MAP_POSITIVE_Z,
GL_TEXTURE_CUBE_MAP_NEGATIVE_Z



# OpenGL

- Load the cube map to the OpenGL context using **setupCubeMap()** in init() function and save the address (e.g. ‘cubemapTexture’)
- **Add** a **function** that draws a skybox.
- Use **`cubeVertices`** as vertex information to define a **`VertexData struct`**.
- Use **DrawVertexArray()** to draw the cube (see GRK exercises).
- Use **glBindTexture**(GL_TEXTURE_CUBE_MAP, cubemapTexture) to bind a cube map texture before drawing the cube

# Vertex Shader

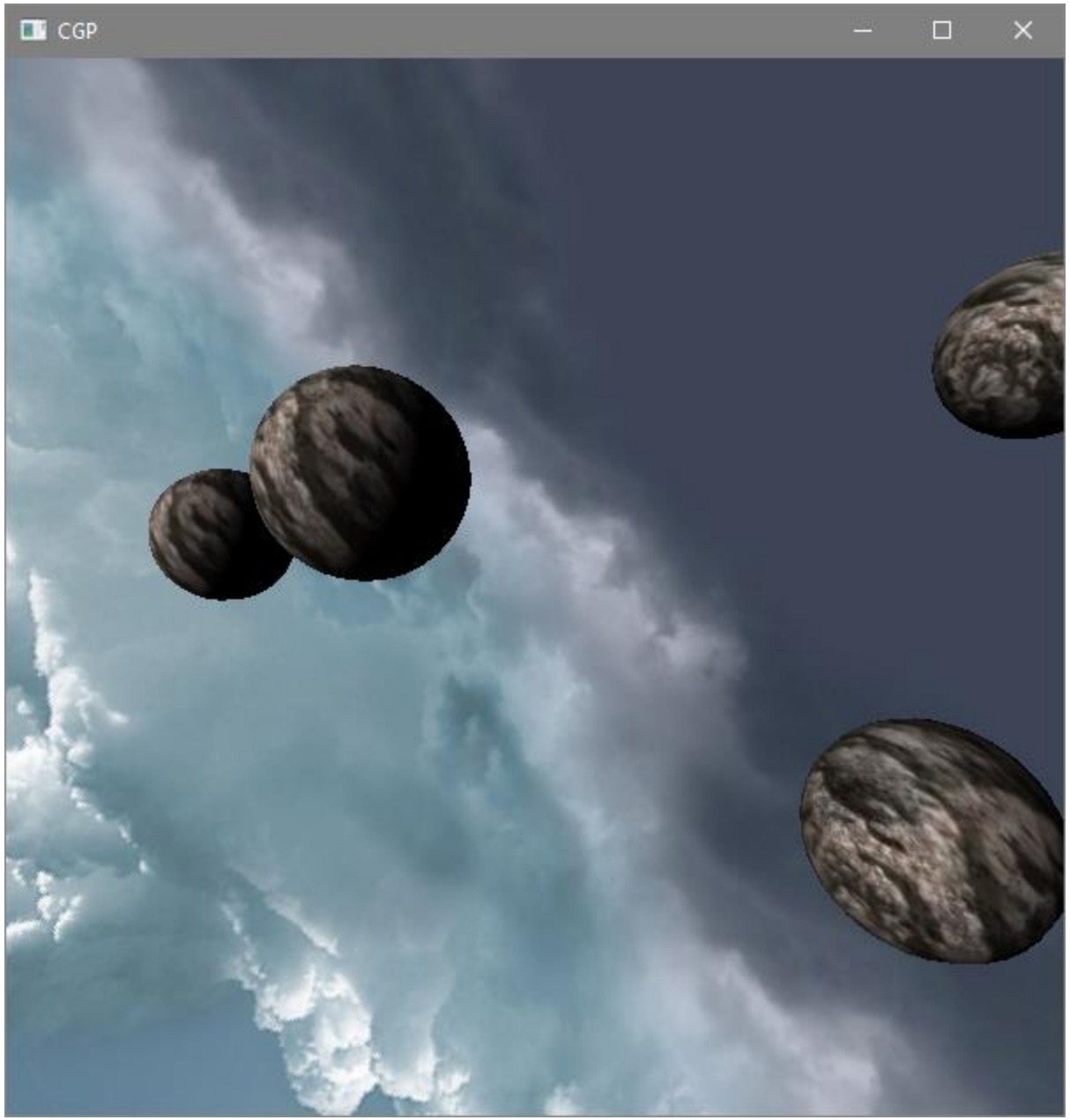
Pass world vertex position as **texture coordinates** to fragment shader

Compute **vertex position** in clip space as usual

# Fragment Shader

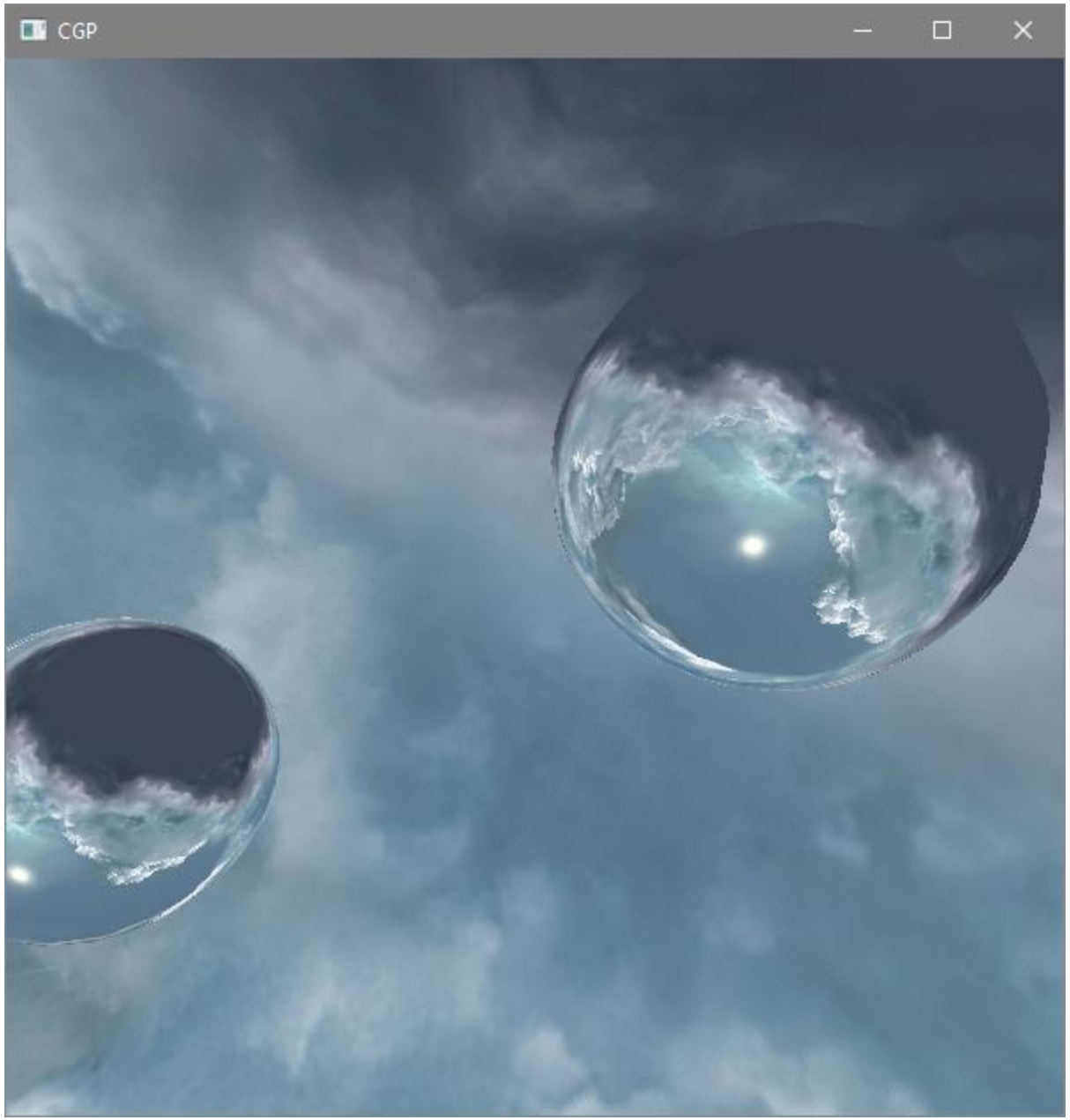
Define **`uniform samplerCube`** for the cube map; give it a name

Access your **cube map texture** (using the **texture**() glsl function and the texture coordinates passed from the vertex shader) to define the **fragment color**


CGP

# Reflection

- OpenGL
  - Apply **cube map** texture to asteroids (use **glBindTexture** again)
- Vertex shader
  - compute **normal** in world space
  - compute **position** in world space
- Fragment shader
  - Pass **camera position** as uniform
  - Calculate **view direction**
  - Compute the **reflection vector** using view and normal vectors
  - Use **reflection vector** instead of texture coordinate to sample the **cube map** texture with texture() function


CGP

# Transparency in OpenGL

glEnable(GL_BLEND);

glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

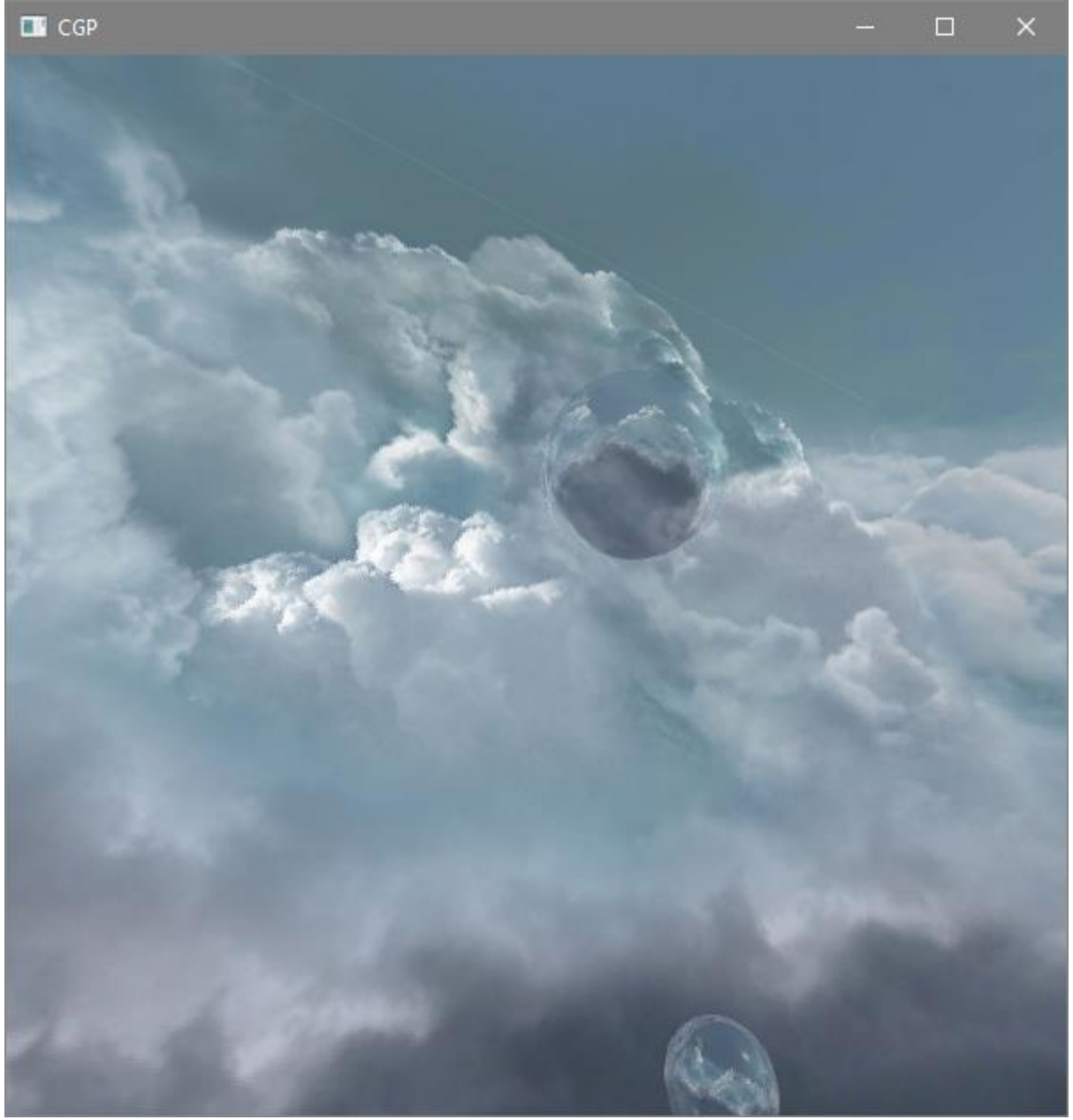
Draw translucent objects

glDisable(GL_BLEND)

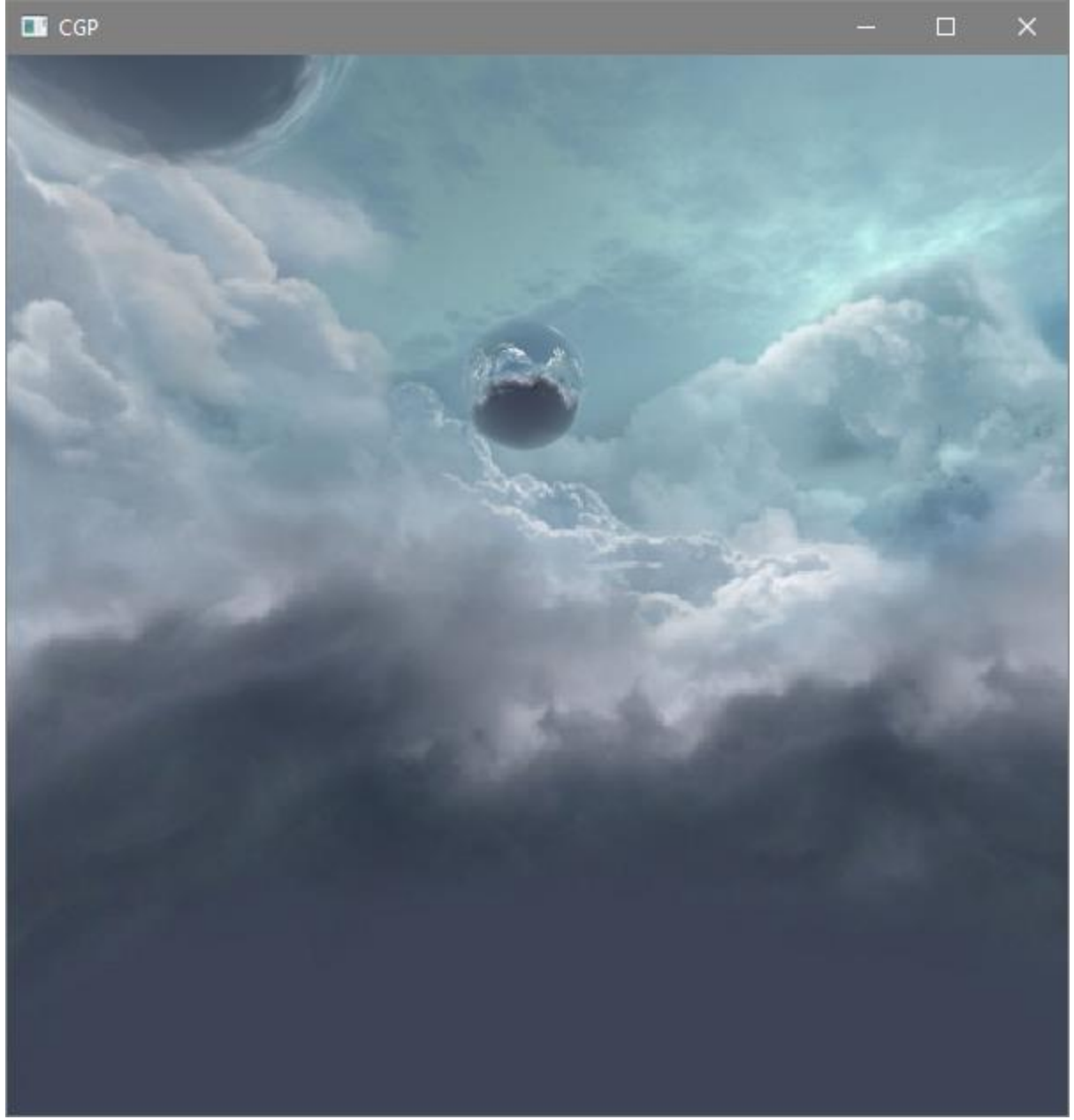
# In shader

glFragColor = vec4(color, opacity);

The variable **opacity** controls the mixture between reflection color and the background

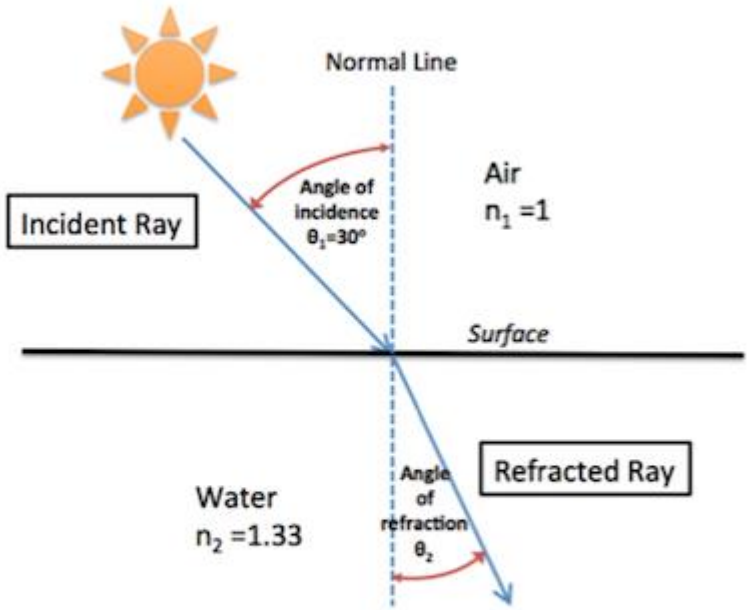



Opacity = 0.5




Opacity = dot(-view, normal)

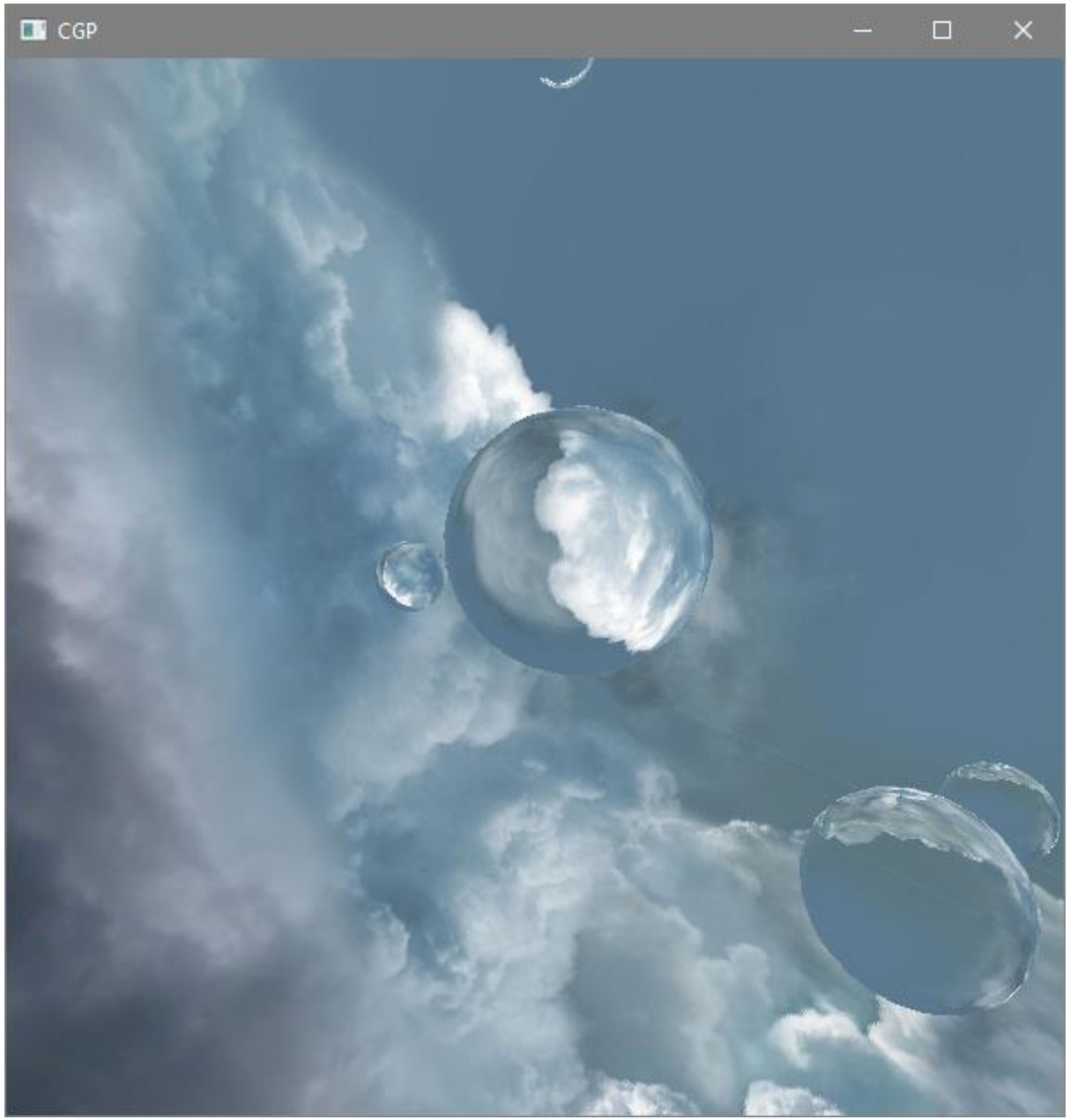
# Refraction


Normal Line
Angle of incidence $\theta_1$=30°
Incident Ray
Air
$n_1$ =1
Surface
Refracted Ray
Angle of refraction $\theta_2$
Water
$n_2$ =1.33

# In shader

Compute refraction vector (use refract(view, n, eta) glsl function where eta is ratio of refractive indices $n_1/n_2$)

Sample cube map texture with refraction vector instead of reflection vector


CGP

# Fresnel effect

Compute reflection vector

Compute refraction vector

Access cube map texture (for both vectors)

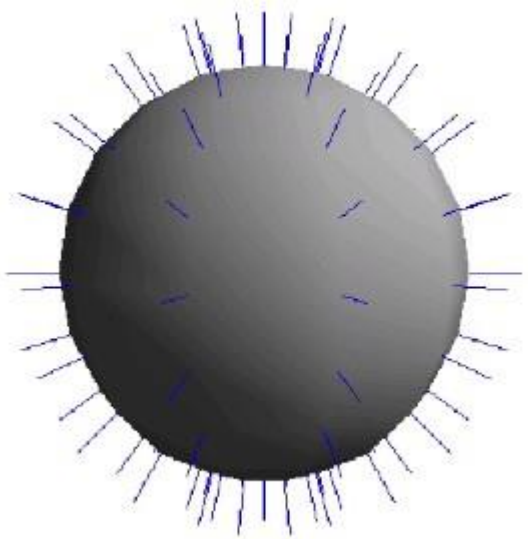
Calculate Schlick approximation

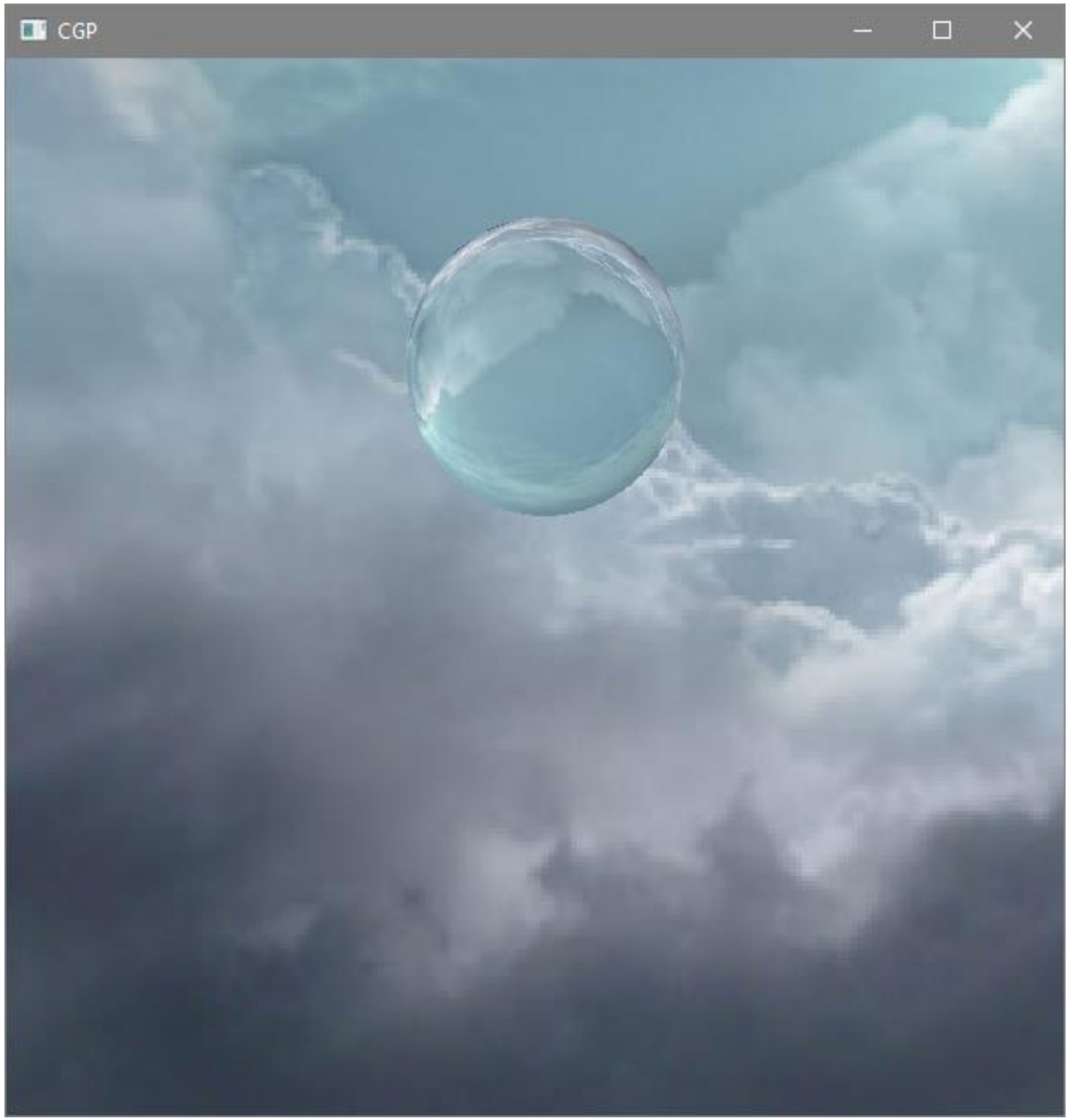
Interpolate color values between reflection and refraction (use mix function: `mix(refracted, reflected, schlick)`)

# Schlick Approximation

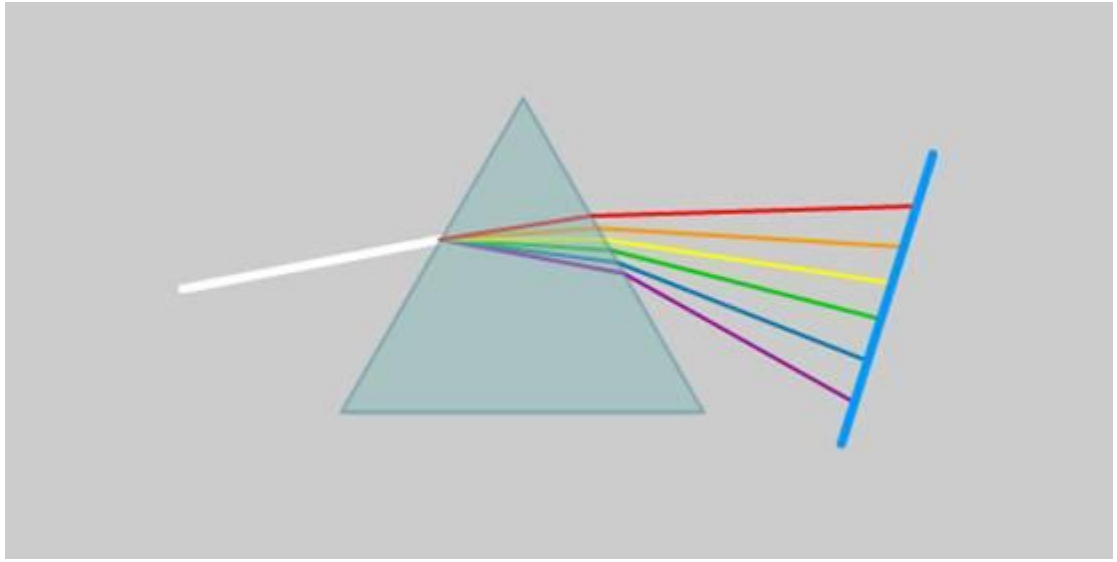
$$f = \frac{\left(1 - \frac{n_1}{n_2}\right)^2}{\left(1 + \frac{n_1}{n_2}\right)^2}$$

$$F = f + (1 - f)\left(1 - \vec{V} \cdot \vec{N}\right)^5$$



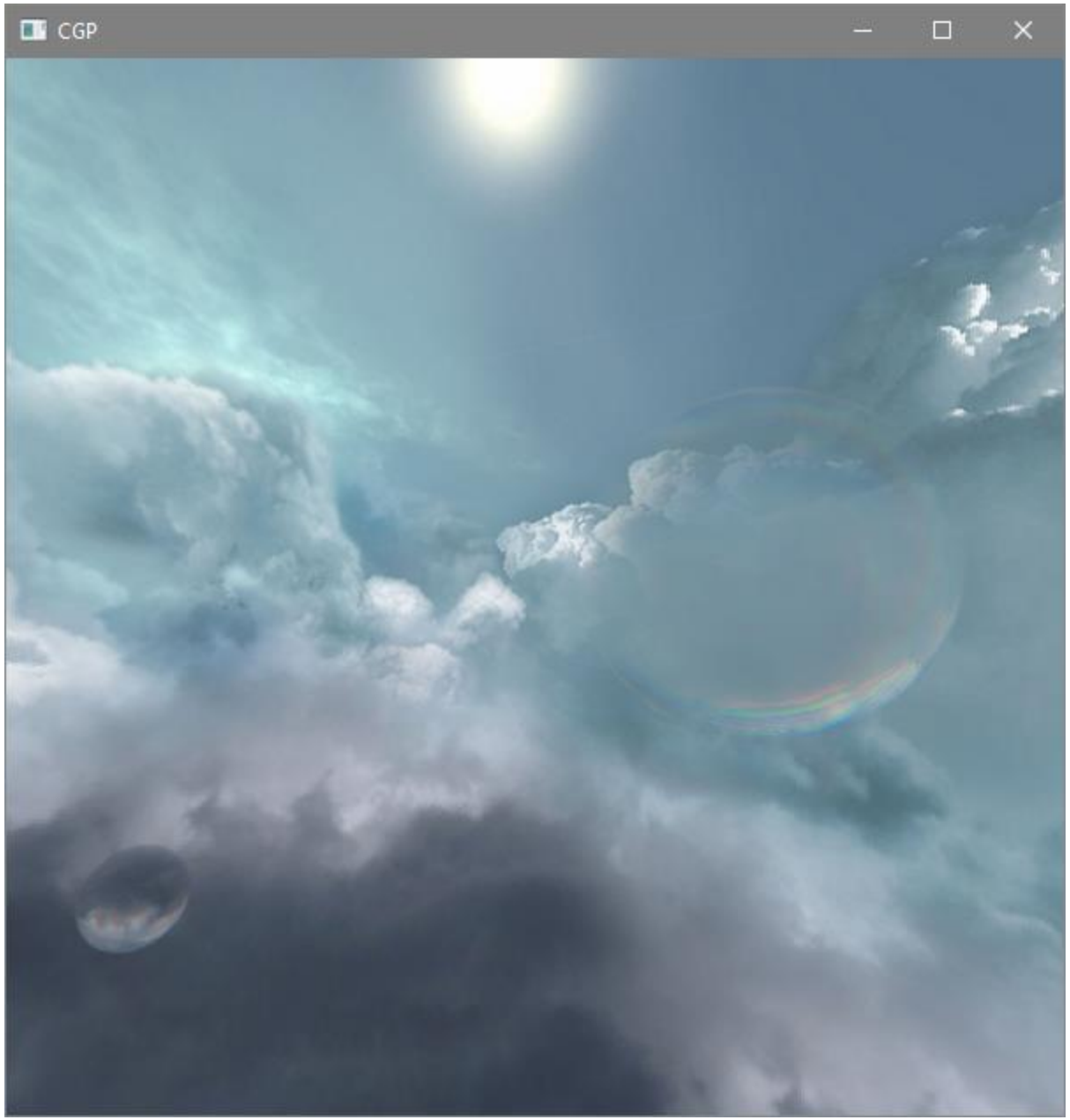

CGP

# Chromatic Dispersion



# Chromatic Dispersion

- Compute refraction for red, green and blue channels separately with a small offstet to n1/n2
- Sample skybox for each color with each calculated value separately to construct the refracted color


CGP

# Try other models…