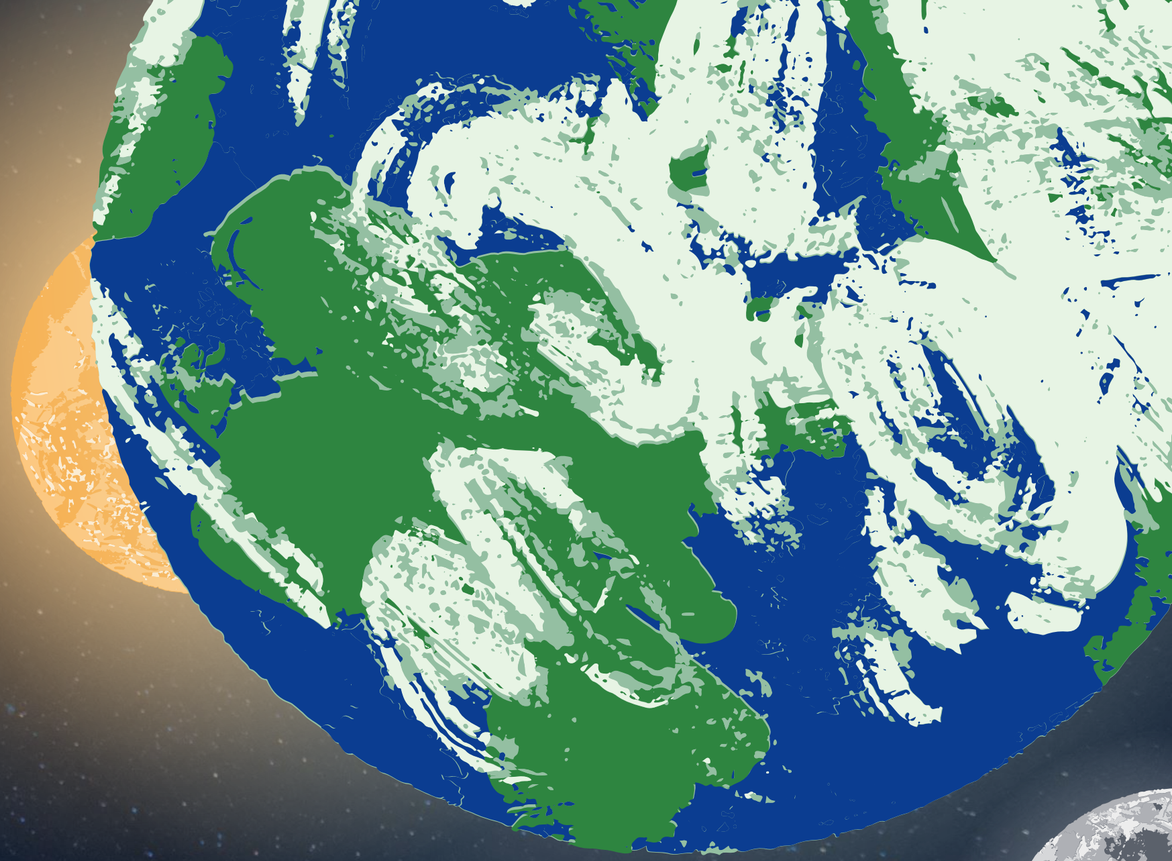
TRABALHO 4

Projecto Final de
Computação Gráfica

SISTEMA SOLAR 3D



Artur Putyato, Gabriel Lázaro, Leandro Serra e Miguel Alexandre

Etapas da Solução

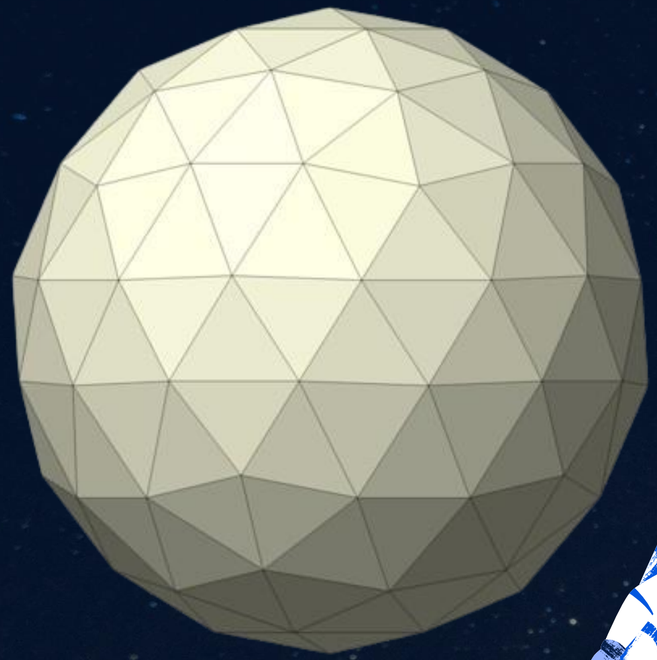
Carregamento de Texturas

Órbitas e Movimentos dos Planetas

Interatividade com o Utilizador

Iluminação da cena

Estrutura Esférica



Definição matemática da Esfera

$$x = r * \cos(a) * \cos(b)$$

$$y = r * \cos(a) * \sin(b)$$

$$z = r * \sin(a)$$

Estrutura Esférica

Em termos de código

```
Sphere(float r, int sectors, int stacks){
        //...

        for (int i = 0; i <= stackCount; ++i)
        {
            stackAngle = (float)(M_PI / 2 - i * stackStep);        // starting from pi/2 to -pi/2
            xy = 1.02f * radius * cosf(stackAngle);             // r * cos(u)
            z = radius * sinf(stackAngle);              // r * sin(u)

            // add (sectorCount+1) vertices per stack
            // the first and last vertices have same position and normal, but different tex coords
            for (int j = 0; j <= sectorCount; ++j)
            {
                sectorAngle = j * sectorStep;           // starting from 0 to 2pi

                // vertex position (x, y, z)
                x = xy * cosf(sectorAngle);             // r * cos(u) * cos(v)
                y = xy * sinf(sectorAngle);             // r * cos(u) * sin(v)
                sphere_vertices.push_back(x);
                sphere_vertices.push_back(y);
                sphere_vertices.push_back(z);

//...
```

Texturas

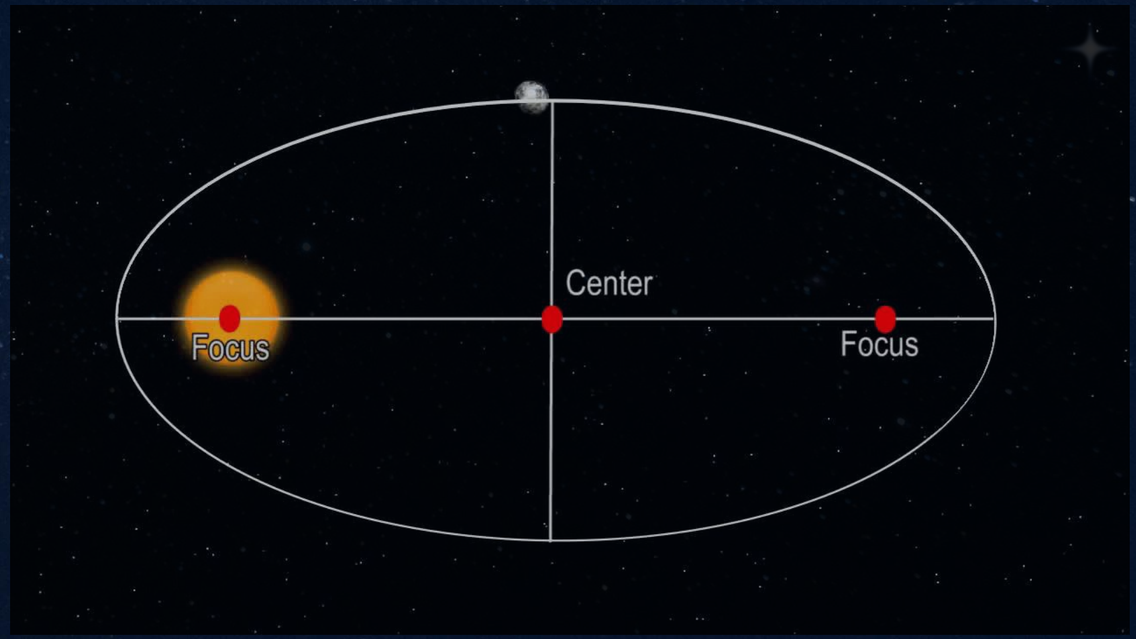
```
GLuint earthTextureID = loadTexture("texturas/earth.jpg");

void setTexture(GLuint textureID, GLuint programID) {
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, textureID);
    glUniform1i(glGetUniformLocation(programID, "myTextureSampler"), 0);
}
```

```
texture(myTextureSampler, UV).xyz;
```

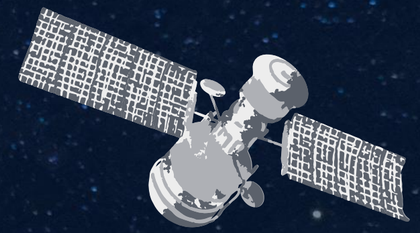
Órbitas no Sistema Solar

Órbitas não são circulos

Fórmulas Elipticas



Órbitas no Sistema Solar

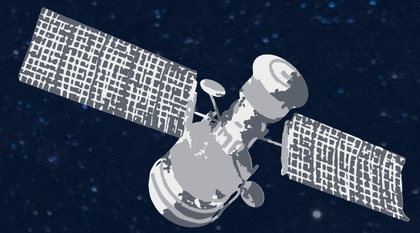


$$R = \frac{R_t \times f \times (1 - e^2)}{1 + e \times \cos\theta}$$

$$\omega = \frac{2\pi}{T}$$

- Função Raio - depende do ângulo
- Velocidade Angular - depende do ano planetar

Órbitas no Sistema Solar



```
auto angular_speed = [speed_factor](double orbit_in_days) -> double {return ((2*3.14159)/orbit_in_days)*speed_factor;};
auto orbitRadius = [pos_earth](double theta, double e, double rad) -> double {return pos_earth*rad*((1.0-e*e)/(1.0+e*std::cos(theta)));};
```

```
angle[2] += angular_speed(365.25);
double radius = orbitRadius(3.14159 * 2 * angle[2] / 360, 0.017, 1);
float x = radius * sin(3.14159 * 2 * angle[2] / 360);
float y = radius * cos(3.14159 * 2 * angle[2] / 360);

glm::mat4 earthModelMatrix = glm::translate(glm::mat4(1.0f), glm::vec3(x, 0.0f, y));
```

Interação com Utilizador

PT INFO

POSIÇÃO DA CÂMERA

```
// Move forward
if (glfwGetKey( window, GLFW_KEY_UP ) == GLFW_PRESS){
    position += direction * deltaTime * speed;}
// Move backward
if (glfwGetKey( window, GLFW_KEY_DOWN ) == GLFW_PRESS){
    position -= direction * deltaTime * speed;}
// Strafe right
if (glfwGetKey( window, GLFW_KEY_RIGHT ) == GLFW_PRESS){
    position += right * deltaTime * speed;}
// Strafe left
if (glfwGetKey( window, GLFW_KEY_LEFT ) == GLFW_PRESS){
    position -= right * deltaTime * speed;}
```

MENU COM INFORMAÇÃO

```
struct PlanetInfo {
    std::string Name;
    std::string OrbitSpeed;
    std::string Mass;
    std::string Gravity;
};


void ShowInfo(GLuint programID2){
RenderText(programID2, <string>, <x>, <y>, <escala>, <corRGB>);
//...
}
```

Iluminação dos planetas

- Componente Difusa:
  - A reflexão difusa representa a luz que é dispersa uniformemente em todas as direções quando incide em uma superfície.
- Componente Especular:
  - A reflexão especular representa o destaque brilhante que ocorre em superfícies quando a luz incide diretamente sobre elas e é refletida de maneira mais concentrada em uma direção específica.
- Componente Ambiente:
  - A reflexão ambiente representa a luz que é dispersa de forma difusa em todas as direções, mas de maneira uniforme, independentemente da posição relativa da fonte de luz.

FRAGMENT SHADER

```
// Ambient
vec3 ambient = ambientStrength * texture(myTextureSampler, UV).xyz;

// Diffuse
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - FragPos);
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * lightColor;

// Specular
vec3 viewDir = normalize(viewPos - FragPos);
vec3 reflectDir = reflect(-lightDir, norm);
float spec = pow(max(dot(viewDir, reflectDir), 0.0), shininess);
vec3 specular = specularStrength * spec * lightColor;

// cor
vec3 result = ambient + diffuse + specular;
```


Planeta: Neptuno
Velocidade Orbital Media (km/s): 5,48
Massa (kg * 10^24): 101.59200
Gravidade (g): 1.17