

解释步骤并给出OpenGL伪代码: 通过关键帧插值动画技术展现出月亮形状从满月到半月再变回满月的变化的过程, 同时体现出亮度的变化



通过关键帧插值动画技术, 我们可以通过计算月亮形状的变化以及亮度的变化, 展现出月亮形状从满月到半月再变回满月的过程。

下面是通过关键帧插值动画技术实现这个效果的伪代码:

SCSS

Copy code

```
// 创建关键帧数组, 记录月亮形状以及亮度的变化 KeyFrame[] keyFrames = { new KeyFrame(0,
shapeFullMoon, brightnessFullMoon), new KeyFrame(0.5, shapeHalfMoon,
brightnessHalfMoon), new KeyFrame(1, shapeFullMoon, brightnessFullMoon) }; // 定义当前时间 float time = 0; // 每帧更新形状和亮度 while (time < 1) { // 计算当前帧的形状和亮度
MoonShape shape = InterpolateShape(keyFrames, time); float brightness =
InterpolateBrightness(keyFrames, time); // 使用当前形状和亮度绘制月亮 DrawMoon(shape,
brightness); // 更新时间 time += deltaTime; }
```

上面的代码假设有一个函数 **InterpolateShape** 可以计算出当前帧的形状, 以及一个函数 **InterpolateBrightness** 可以计算出当前帧的亮度, 并且假设有一个函数 **DrawMoon** 可以根据当前形状和亮度绘制月亮。



解释步骤并给出OpenGL伪代码: 通过路径动画技术, 给出鹊桥 (路径), 牛郎织女分别在桥两头走到一起, 可以是图形方式, 也可以是图像方式



通过路径动画技术, 我们可以让牛郎织女在鹊桥两头沿着预定义的路径走到一起。

下面是通过路径动画技术实现这个效果的伪代码:

SCSS

Copy code

```
// 定义鹊桥路径 Path path = GeneratePath(start1, end1, start2, end2); // 定义牛郎织女图形
Shape cowboy = GenerateShape("cowboy"); Shape weaver = GenerateShape("weaver"); // 定义当前时间 float time = 0; // 每帧更新牛郎织女位置 while (time < 1) { // 计算当前帧牛郎织女的位置
vec2 cowboyPos = InterpolatePath(path, time / 2); vec2 weaverPos =
InterpolatePath(path, (time + 1) / 2); // 使用当前位置绘制牛郎织女图形 DrawShape(cowboy,
cowboyPos); DrawShape(weaver, weaverPos); // 更新时间 time += deltaTime; }
```

如果你想使用图像代替图形，你可以把上面的代码中的 `GenerateShape` 和 `DrawShape` 替换为 `LoadImage` 和 `DrawImage`。

注意，上面的代码假设有一个函数 `GeneratePath` 可以生成鹊桥路径，以及一个函数 `InterpolatePath` 可以计算出当前帧的位置，并假设有一个函数 `DrawShape` 或 `DrawImage` 可以绘制图形或图像。



17

解释步骤并给出OpenGL伪代码: 通过变形动画算法, 完成牛郎织女互变（渐变动画），可以是图形方式，也可以是图像方式



通过变形动画算法，我们可以实现牛郎织女互相变换，从而实现一个渐变动画效果。

以下是通过变形动画算法实现牛郎织女互相变换的伪代码：

SCSS

Copy code

```
// 定义牛郎织女图形 Shape cowboy = GenerateShape("cowboy"); Shape weaver =  
GenerateShape("weaver"); // 定义当前时间 float time = 0; // 每帧更新牛郎织女图形 while  
(time < 1) { // 计算当前帧牛郎织女图形的变形参数 float t = Interpolate(time, 0, 1); mat4  
cowboyTransform = MixTransforms(IdentityMatrix(), GetWeaverTransform(), t); mat4  
weaverTransform = MixTransforms(GetCowboyTransform(), IdentityMatrix(), t); // 使用当前  
变形参数绘制牛郎织女图形 DrawShape(cowboy, cowboyTransform); DrawShape(weaver,  
weaverTransform); // 更新时间 time += deltaTime; }
```

如果你想使用图像代替图形，你可以把上面的代码中的 `GenerateShape` 和 `DrawShape` 替换为 `LoadImage` 和 `DrawImage`。

注意，上面的代码假设有一个函数 `Interpolate` 可以根据当前时间计算出变形参数，以及一个函数 `MixTransforms` 可以将两个变形参数按照指定比例混合，并假设有一个函数 `GetCowboyTransform` 和 `GetWeaverTransform` 可以获得牛郎和织女的初始变形参数。



17

解释步骤并给出OpenGL伪代码: 通过角色动画算法, 给出牛郎织女两人分别挥手告别, 可以是图形方式，也可以是图像方式



通过角色动画算法，我们可以实现牛郎织女挥手告别的动画效果。

以下是通过角色动画算法实现牛郎织女挥手告别的伪代码：

```
// 定义牛郎织女图形 Shape cowboy = GenerateShape("cowboy"); Shape weaver =
GenerateShape("weaver"); // 定义牛郎织女动画数据 AnimationData cowboyAnimation =
LoadAnimation("cowboy_wave"); AnimationData weaverAnimation =
LoadAnimation("weaver_wave"); // 定义当前时间 float time = 0; // 每帧更新牛郎织女图形
while (time < cowboyAnimation.duration && time < weaverAnimation.duration) { // 计算当
前帧牛郎织女图形的变形参数 mat4 cowboyTransform = EvaluateAnimation(cowboyAnimation,
time); mat4 weaverTransform = EvaluateAnimation(weaverAnimation, time); // 使用当前变形
参数绘制牛郎织女图形 DrawShape(cowboy, cowboyTransform); DrawShape(weaver,
weaverTransform); // 更新时间 time += deltaTime; }
```

如果你想使用图像代替图形，你可以把上面的代码中的 `GenerateShape` 和 `DrawShape` 替换为 `LoadImage` 和 `DrawImage`。

注意，上面的代码假设有一个函数 `LoadAnimation` 可以加载动画数据，以及一个函数 `EvaluateAnimation` 可以根据当前时间计算出当前帧的变形参数。



17

解释步骤并给出OpenGL伪代码: 通过粒子动画算法, 给出烟花庆祝或闪闪星光或流星飞过, 或祥云飘飘



步骤:

1. 创建一系列粒子，每一个粒子代表一个星星或烟花的火花。
2. 初始化每一个粒子的位置、速度、大小等参数。
3. 在每一帧的动画中，更新每一个粒子的位置，根据其速度和加速度。
4. 在每一帧的动画中，根据粒子的位置和大小绘制它们。

OpenGL伪代码:

```
# 定义粒子结构体, 包含位置、速度、大小等参数 struct Particle { vec3 position; vec3
velocity; float size; }; # 创建粒子数组 Particle particles[N]; # 初始化粒子数组 void
initParticles() { for (int i = 0; i < N; i++) { particles[i].position = vec3(...);
particles[i].velocity = vec3(...); particles[i].size = ...; } } # 更新粒子数组 void
updateParticles() { for (int i = 0; i < N; i++) { particles[i].position +=
particles[i].velocity; particles[i].velocity += vec3(...); } } # 绘制粒子数组 void
drawParticles() { for (int i = 0; i < N; i++) { glPointSize(particles[i].size);
glBegin(GL_POINTS); glVertex3f(particles[i].position.x, particles[i].position.y,
particles[i].position.z); glEnd(); } }
```

Q1

```
1. // 创建关键帧数组，记录月亮形状以及亮度的变化
2. KeyFrame[] keyFrames = {
3.     new KeyFrame(0, shapeFullMoon, brightnessFullMoon),
4.     new KeyFrame(0.5, shapeHalfMoon, brightnessHalfMoon),
5.     new KeyFrame(1, shapeFullMoon, brightnessFullMoon)
6. };
7.
8. // 定义当前时间
9. float time = 0;
10.
11. // 每帧更新形状和亮度
12. while (time < 1) {
13.     // 计算当前帧的形状和亮度
14.     MoonShape shape = InterpolateShape(keyFrames, time);
15.     float brightness = InterpolateBrightness(keyFrames, time);
16.
17.     // 使用当前形状和亮度绘制月亮
18.     DrawMoon(shape, brightness);
19.
20.     // 更新时间
21.     time += deltaTime;
22. }
```

Q2

```
1. // 定义鹊桥路径
2. Path path = GeneratePath(start1, end1, start2, end2);
3.
4. // 定义牛郎织女图形
5. Shape cowboy = GenerateShape("cowboy");
6. Shape weaver = GenerateShape("weaver");
7.
8. // 定义当前时间
9. float time = 0;
10.
11. // 每帧更新牛郎织女位置
12. while (time < 1) {
13.     // 计算当前帧牛郎织女的位置
14.     vec2 cowboyPos = InterpolatePath(path, time / 2);
15.     vec2 weaverPos = InterpolatePath(path, (time + 1) / 2);
16.
17.     // 使用当前位置绘制牛郎织女图形
18.     DrawShape(cowboy, cowboyPos);
19.     DrawShape(weaver, weaverPos);
20.
21.     // 更新时间
22.     time += deltaTime;
23. }
```

Q3

```
1. // 定义牛郎织女图形
2. Shape cowboy = GenerateShape("cowboy");
3. Shape weaver = GenerateShape("weaver");
4.
5. // 定义当前时间
6. float time = 0;
7.
8. // 每帧更新牛郎织女图形
9. while (time < 1) {
10.    // 计算当前帧牛郎织女图形的变形参数
11.    float t = Interpolate(time, 0, 1);
12.    mat4 cowboyTransform = MixTransforms(IdentityMatrix(), GetWeaverTransform(), t);
13.    mat4 weaverTransform = MixTransforms(GetCowboyTransform(), IdentityMatrix(), t);
14.
15.    // 使用当前变形参数绘制牛郎织女图形
16.    DrawShape(cowboy, cowboyTransform);
17.    DrawShape(weaver, weaverTransform);
18.
19.    // 更新时间
20.    time += deltaTime;
21. }
```

Q4

```
1. // 定义牛郎织女图形
2. Shape cowboy = GenerateShape("cowboy");
3. Shape weaver = GenerateShape("weaver");
4.
5. // 定义牛郎织女动画数据
6. AnimationData cowboyAnimation = LoadAnimation("cowboy_wave");
7. AnimationData weaverAnimation = LoadAnimation("weaver_wave");
8.
9. // 定义当前时间
10. float time = 0;
11.
12. // 每帧更新牛郎织女图形
13. while (time < cowboyAnimation.duration && time < weaverAnimation.dura
    tion) {
14.     // 计算当前帧牛郎织女图形的变形参数
15.     mat4 cowboyTransform = EvaluateAnimation(cowboyAnimation, time);
16.     mat4 weaverTransform = EvaluateAnimation(weaverAnimation, time);
17.
18.     // 使用当前变形参数绘制牛郎织女图形
19.     DrawShape(cowboy, cowboyTransform);
20.     DrawShape(weaver, weaverTransform);
21.
22.     // 更新时间
23.     time += deltaTime;
24. }
```

Q5

```
1. # 定义粒子结构体, 包含位置、速度、大小等参数
2. struct Particle {
3.     vec3 position;
4.     vec3 velocity;
5.     float size;
6. };
7.
8. # 创建粒子数组
9. Particle particles[N];
10.
11. # 初始化粒子数组
12. void initParticles() {
13.     for (int i = 0; i < N; i++) {
14.         particles[i].position = vec3(...);
15.         particles[i].velocity = vec3(...);
16.         particles[i].size = ...;
17.     }
18. }
19.
20. # 更新粒子数组
21. void updateParticles() {
22.     for (int i = 0; i < N; i++) {
23.         particles[i].position += particles[i].velocity;
24.         particles[i].velocity += vec3(...);
25.     }
26. }
27.
28. # 绘制粒子数组
29. void drawParticles() {
30.     for (int i = 0; i < N; i++) {
31.         glPointSize(particles[i].size);
32.         glBegin(GL_POINTS);
33.         glVertex3f(particles[i].position.x, particles[i].position.y,
34.             particles[i].position.z);
35.         glEnd();
36.     }
```