**《计算机图形学》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **年级、专业、班级** | | **22计算机科学与技术（卓越）01班** | | | **姓名** |  |
| **实验题目** | 实验四 路径追踪 | | | | | |
| **实验时间** | **25.4.28** | | **实验地点** | **DS305** | | |
| **实验成绩** |  | | **实验性质** | **□验证性** √**设计性 □综合性** | | |
| 教师评价：  □算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；  □实验结果正确； □语法、语义正确； □报告规范；  其他：  评价教师签名： | | | | | | |
| 实验目的  1. 实现完整的Path Tracing算法  2. 通过Path Tracing算法的实现，锻炼编程实践能力，加深对理论知识的理解 | | | | | | |
| 二、实验项目内容  1.这次主要完成Path-Tracing部分  2.提高部分应用到了材质和外观的知识。 | | | | | | |
| 1. 实验过程或算法（源程序）   **Microfacet：**  在main.cpp中添加Material并处理输入添加box:   1. Material\* mat\_dif = new Material(DIFFUSE, Vector3f(0.0f)); 2. mat\_dif->Kd = Vector3f(0.725f, 0.71f, 0.68f); 3. Material\* mat\_mic = new Material(MICROFACET, Vector3f(0.0f)); 4. mat\_mic->Kd = Vector3f(0.0f, 0.0f, 0.0f); 5. mat\_mic->ior = 40.0f;*// 折射率* 6. Material\* light = new Material(DIFFUSE, (8.0f \* Vector3f(0.747f+0.058f, 0.747f+0.258f, 0.747f) + 15.6f \* Vector3f(0.740f+0.287f,0.740f+0.160f,0.740f) + 18.4f \*Vector3f(0.737f+0.642f,0.737f+0.159f,0.737f))); 7. light->Kd = Vector3f(0.65f); 8. *// 根据输出设置* 9. Material\* box\_mat = mat\_dif; 10. if (argc >= 1) 11. { 12. if (!strcmp(argv[1], "MICROFACET")) 13. box\_mat = mat\_mic; 14. } 15. MeshTriangle shortbox("../models/cornellbox/shortbox.obj", box\_mat); 16. MeshTriangle tallbox("../models/cornellbox/tallbox.obj", box\_mat);   之后再在Material.hpp中添加方向采样和概率密度函数：   1. Vector3f Material::sample(const Vector3f &wi, const Vector3f &N){ 2. switch(m\_type){ 3. case DIFFUSE: 4. case MICROFACET: 5. { 6. *// 均匀对半球采样 z[0,1] phi[0,2\pi]* 7. float x\_1 = get\_random\_float(), x\_2 = get\_random\_float();*// x\_1->z x\_2->phi \in [0,1]random* 8. float z = std::fabs(1.0f - 2.0f \* x\_1);*// z[0,1]* 9. float r = std::sqrt(1.0f - z \* z), phi = 2 \* M\_PI \* x\_2;*// 垂直N平面* 10. Vector3f localRay(r\*std::cos(phi), r\*std::sin(phi), z); 11. return toWorld(localRay, N); 12. break; 13. } 14. } 15. } 16. float Material::pdf(const Vector3f &wi, const Vector3f &wo, const Vector3f &N){ 17. switch(m\_type){ 18. case DIFFUSE: 19. case MICROFACET: 20. { 21. *// uniform sample probability 1 / (2 \* PI)* 22. if (dotProduct(wo, N) > 0.0f) 23. return 0.5f / M\_PI; 24. else 25. return 0.0f; 26. break; 27. } 28. } 29. }   **微表面漫反射BRDF**  IMG_256   1. *// 法线分布函数D* 2. float DistributionGGX(Vector3f n, Vector3f h, float roughness) 3. { 4. float a = roughness \* roughness, a2 = a\*a; 5. float nh = std::max(dotProduct(n, h), 0.0f), nh2 = nh \* nh; 6. float div = (nh2 \* (a2 - 1.0) + 1.0); 7. div = M\_PI \* div \* div; 8. return a2 / std::max(div, eps);*// 防止分母为0* 9. }   **几何分布函数：**  *// 几何分布函数G*   1. float geometrySchlickGGX(float nv, float k) 2. { 3. float div = nv \* (1.0 - k) + k; 4. return nv / div; 5. } 6. float geometrySmith(Vector3f n, Vector3f v, Vector3f l, float roughness) 7. { 8. float r = (roughness + 1.0); 9. float k = (r \* r) / 8.0; 10. float nv = std::max(dotProduct(n, v), 0.0f); 11. float nl = std::max(dotProduct(n, l), 0.0f); 12. float ggx2 = geometrySchlickGGX(nv, k); 13. float ggx1 = geometrySchlickGGX(nl, k); 14. return ggx1 \* ggx2; 15. }   **菲涅尔函数F：**   1. *// 法线分布函数D* 2. float distributionGGX(Vector3f n, Vector3f h, float roughness) 3. { 4. float a = roughness \* roughness, a2 = a\*a; 5. float nh = std::max(dotProduct(n, h), 0.0f), nh2 = nh \* nh; 6. float div = (nh2 \* (a2 - 1.0) + 1.0); 7. div = M\_PI \* div \* div; 8. return a2 / std::max(div, eps);*// 防止分母为0* 9. }   **下面在Material::eval中应用镜面反射BRDF：**   1. Vector3f Material::eval(const Vector3f &wi, const Vector3f &wo, const Vector3f &N){ 2. switch(m\_type){ 3. case DIFFUSE: 4. { 5. *// calculate the contribution of diffuse   model* 6. float cosalpha = dotProduct(N, wo); 7. if (cosalpha > 0.0f) { 8. Vector3f diffuse = Kd / M\_PI; 9. return diffuse; 10. } 11. else 12. return Vector3f(0.0f); 13. break; 14. } 15. case MICROFACET: 16. { 17. float cosa = dotProduct(N, wo); 18. if (cosa > 0.0f) 19. { 20. float roughness = 0.4f; 21. Vector3f v = -wi, l = wo, h = normalize(v + l), n = N; 22. float D = distributionGGX(n, h, roughness); 23. float G = geometrySmith(n, v, l, roughness); 24. float F; 25. float etat = 1.85;*// 介质折射率* 26. fresnel(wi, n, etat, F); 27. Vector3f div1 = D \* G \* F; 28. float div2 = 4 \* std::max(dotProduct(n, v), 0.0f) \* std::max(dotProduct(n, l), 0.0f); 29. Vector3f fr = div1 / std::max(div2, eps);*// 防止div0* 30. *// 能量守恒* 31. float ks\_ = F;*// 反射比率* 32. float kd\_ = 1.0f - ks\_;*// 折射比率* 33. Vector3f diffuse = 1.0f / M\_PI; 34. *// 因为在 specular 项里已经考虑了反射部分的比例：F。所以反射部分不需要再乘以 ks\_* 35. *// Ks为镜面反射项，Kd为漫反射项。* 36. return Ks \* fr + kd\_ \* Kd \* diffuse; 37. } 38. else 39. return Vector3f(0.0f); 40. break; 41. } 42. } 43. }   **添加球：**  首先更改sphere.hpp中对球相交判定的精度问题，若不修改，程序在运行中会报错，因为当t0=0时，有可能会造成除0错误。   1. Intersection getIntersection(Ray ray){ 2. Intersection result; 3. result.happened = false; 4. Vector3f L = ray.origin - center; 5. float a = dotProduct(ray.direction, ray.direction); 6. float b = 2 \* dotProduct(ray.direction, L); 7. float c = dotProduct(L, L) - radius2; 8. float t0, t1; 9. if (!solveQuadratic(a, b, c, t0, t1)) return result; 10. if (t0 < 0) t0 = t1; 11. if (t0 < 0) return result; 12. *// 相交判定修改* 13. if (t0 > 0.5) { 14. result.happened = true; 15. result.coords = Vector3f(ray.origin + ray.direction \* t0); 16. result.normal = normalize(Vector3f(result.coords - center)); 17. result.m = this->m; 18. result.obj = this; 19. result.distance = t0; 20. } 21. return result; 22. } 23. Vector3f evalDiffuseColor(const Vector2f &st)const { 24. *//return m->getColor();* 25. return {}; 26. }   **最后在main.cpp中添加Sphere：**   1. Material\* mat\_dif = new Material(DIFFUSE, Vector3f(0.0f)); 2. mat\_dif->Kd = Vector3f(0.725f, 0.71f, 0.68f); 3. Material\* mat\_mic = new Material(MICROFACET, Vector3f(0.0f)); 4. mat\_mic->Ks = Vector3f(0.45, 0.45, 0.45); 5. mat\_mic->Kd = Vector3f(0.3, 0.3, 0.25); 6. Material\* light = new Material(DIFFUSE, (8.0f \* Vector3f(0.747f+0.058f, 0.747f+0.258f, 0.747f) + 15.6f \* Vector3f(0.740f+0.287f,0.740f+0.160f,0.740f) + 18.4f \*Vector3f(0.737f+0.642f,0.737f+0.159f,0.737f))); 7. light->Kd = Vector3f(0.65f); 8. *// 根据输出设置* 9. bool loadbox = 1; 10. Material\* box\_mat = mat\_dif; 11. if (argc >= 1) 12. { 13. if (!strcmp(argv[1], "MICROFACET") || !strcmp(argv[1], "SPHERE")) 14. { 15. box\_mat = mat\_mic; 16. if (!strcmp(argv[1], "SPHERE")) 17. loadbox = 0; 18. } 19. } 20. MeshTriangle shortbox("../models/cornellbox/shortbox.obj", box\_mat); 21. MeshTriangle tallbox("../models/cornellbox/tallbox.obj", box\_mat); 22. Sphere sphere(Vector3f(150, 100, 300), 100, mat\_mic); 23. if (loadbox) 24. { 25. scene.Add(&shortbox); 26. scene.Add(&tallbox); 27. } 28. else 29. scene.Add(&sphere); | | | | | | |
| 1. 实验结果及分析和（或）源程序调试过程   经过运行代码可以成功得到以下图案，证明实验结果正确：    遇到的问题： 输出偏暗有横线  解决方法：如果有渲染结果较暗，出现横向黑色条纹的情况，那么，很可能是因为直接光部分由于精度问题，被错误判断为遮挡，可以试着通过精度调整放宽相交限制（将EPSILON变量增大），并且浮点数相等可能性很低，所以在判断条件中设置为 t\_enter < t\_exit而忘了等号，所以可能会有一定的出入 | | | | | | |