Assignment 3:
Ray Tracing with Direct Lighting

NAME: YANG HONGDI
STUDENT NUMBER: 2019533234
EMAIL: YANGHD@SHANGHAITECH.EDU.CN

1 INTRODUCTION

In this project, simple Ray tracing with Direct lighting is performed. Phong lighting model is used for radiance calculation. Anti-aliasing is performed with rotated grid pattern. Some textures have been added to the Boxes with normal mapping.

2 IMPLEMENTATION DETAILS

2.1 Pin-hole camera model

First, we construct axes for camera coordinate system by using lookat point and reference up direction.

```
void Camera::lookAt(const vec3 &lookAt, const
     vec3 &refUp) {
  this->forward = (position - lookAt).
      normalized();
  this->right = refUp.cross(forward).
      normalized();
  this->up = forward.cross(right).normalized
      ();
  this->halfFovScale = tan(radians(
      verticalFov * 0.5f));
}
```

Then we can generate ray from camera for a specified pixel.

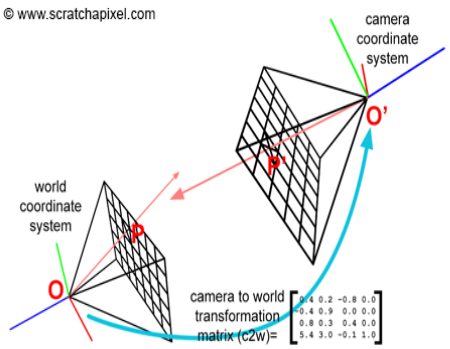


Fig. 1. world space coordinate calculation

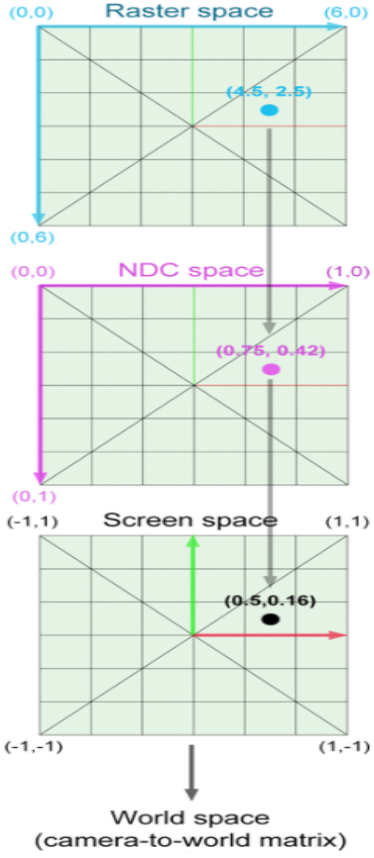


Fig. 2. coordinate transform

As the image shows, we already have the camera coordinate system, by simply transforming the pixel coordinate in raster space to coordinate in screen space, then we can easily compute its position in world space.
Then we shoot a ray from the camera position to pixel position, and we set its origin as the pixel position and the direction as $norm(P_{pixel} - P_{camera})$

```
Ray Camera::generateRay(Float dx, Float dy)
    const {
  vec3 position;
  vec3 dir;

  Float width = film.resolution.x();
  Float height = film.resolution.y();
```

```
  Float screen_x, screen_y;
  screen_x = (2 * (dx)/width - 1) * film.
      getAspectRatio() * halfFovScale *
      focalLength;
  screen_y = (2 * (dy)/height - 1) *
      halfFovScale * focalLength;
  vec3 screenCenter = this->position -
      focalLength * forward;
  position = screenCenter + screen_x * right
      + screen_y * up;
  dir = (position - this->position).
      normalized();

return Ray{position, dir};
}
```

2.2 Ray-triangle intersection test

To test if the ray intersects with the triangle, we adopt Moller-Turmbore instersection algorithm.
First, we know the ray's parametric equation :

$$P = O + tD$$

And we can denot any point in the triangle as :

$$P = (1 - u - v)A + uB + uC$$

so we have :

$$O + tD = (1 - u - v)A + uB + uC$$

by some transformation :

$$\begin{bmatrix} -D & (B-A) & (C-A) \end{bmatrix} \begin{bmatrix} t \\ u \\ v \end{bmatrix} = O - A$$

We denote $T = O - A, E_1 = B - A$ and $E_2 = C - A$, applying Cramer's rule and determinant property, we get :

$$\begin{bmatrix} t \\ u \\ v \end{bmatrix} = \frac{1}{P \cdot E_1} \begin{bmatrix} Q \cdot E_2 \\ P \cdot T \\ Q \cdot D \end{bmatrix}$$

where $P = D \times E_2$ and $D = T \times E_1$. Now we can easily compute t, u, v Here are the code.

```
bool Triangle::intersect(Interaction &
    interaction, const Ray &ray) const {
  const vec3 &v0 = mesh->p[v[0]];
  const vec3 &v1 = mesh->p[v[1]];
  const vec3 &v2 = mesh->p[v[2]];

  vec3 E1 = v1 - v0;
  vec3 E2 = v2 - v0;
  vec3 T = ray.origin - v0;

  vec3 P = ray.direction.cross(E2);
  vec3 Q = T.cross(E1);
  Float det = P.dot(E1);
  if (abs(det) < SHADOW_EPS) return false;
  vec3 tuv = 1/(P.dot(E1)) * vec3(Q.dot(E2),
      P.dot(T), Q.dot(ray.direction));
  Float t = tuv.x();
  Float u = tuv.y();
  Float v = tuv.z();
  ...
}
```

And when $t \geq 0, u \geq 0, v \geq 0, 1 - u - v \geq 0$, the ray intersects with the triangle.

2.3 Area Light

For area light, we just uniformly sample some point lights on it. And for every point light, the radiance will be $\frac{L}{N}$, where N is the number of samples.

2.4 Phong lighting integrator

To calculate the irrandiance, we use phong lighting model. Basically we have following steps :

1. For each ray, we check how it intersects with the scene.
   - If the ray intersects with nothing, return (0,0,0).
   - If the ray intersects with light, return light's color.
   - If the ray intersects with geometry, go to step 2.
2. We shoot another ray from the intersection point to one light sample.
   - If the ray is blocked by other objects, we only add ambient light.
   - If the ray only intersects with the light, we calculate the phong lighting on that point.
3. We repeat step2 for every light sample on area light.

Then we have the irrandiance, here are the code.

```
vec3 PhongLightingIntegrator::radiance(Scene
    &scene,
            const Interaction &interaction,
            const Ray &ray) const {
  if (interaction.type == interaction.NONE)
  {
    return vec3::Zero();
  }
  else if (interaction.type == interaction.
      LIGHT)
  {
    return scene.getLight()->getColor();
  }
  else if (interaction.type == interaction.
      GEOMETRY)
  {
    vec3 ambient = (scene.getAmbientLight().
        array() * interaction.lightingModel.
        ambient.array()).matrix();
    vec3 diffuse = vec3::Zero();
    vec3 specular = vec3::Zero();
```

```
    for (LightSamplePair pointLight : scene.
        getLight()->samples())
    {
      ///Diffuse
      vec3 lightDir = (pointLight.first -
          interaction.entryPoint).normalized
          ();
      Ray lightRay(interaction.entryPoint,
          lightDir);
      if (!scene.isShadowed(lightRay))
      {
        ///Diffuse
        Float diff = std::max(interaction.
            normal.dot(lightDir), 0.0f);
        Eigen::Array3f temp_diff = diff *
            pointLight.second.array() *
            interaction.lightingModel.
            diffusion.array();
        diffuse += temp_diff.matrix();

        ///Specluar
        vec3 reflectDir = (2 * lightDir.dot(
            interaction.normal) * interaction
            .normal - lightDir).normalized();
        Float spec = (Float)std::pow(std::max
            (reflectDir.dot(-ray.direction),
            0.0f), interaction.lightingModel.
            shininess);
        Eigen::Array3f temp_spec =  spec *
            pointLight.second.array() *
            interaction.lightingModel.
            specular.array();
        specular += temp_spec.matrix();
      }
    }
    return ambient + diffuse + specular;
  }
  return vec3::Zero();
}
```

To render the scene, we just apply the function on every pixel.

2.5 Anti aliasing

For anti aliasing, rotated grid is used for sampling in pixel.

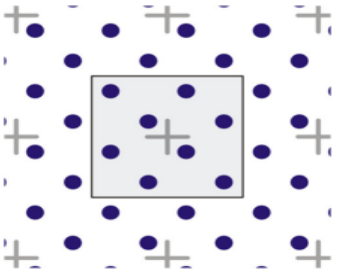


Fig. 3. Rotated grid

To perform anti aliasing and also for time effiency, we sample 4 points in each pixel with a rotation of 26.6°.

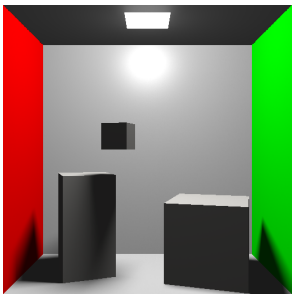
2.6 Texture

Texture with normal mapping is performed. As in section 2.2 Ray-triangle intersection test, we have u, v. So we simply load the color on pixel $(int)\frac{u}{width}, (int)\frac{v}{height}$ on the Texture map.

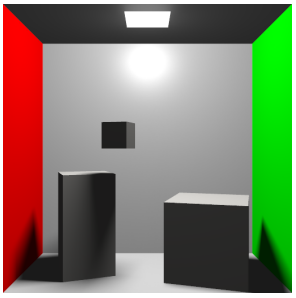
```
vec3 Texture::getColor(vec2 uv) const
{
  int x = (int) (uv.x() * width);
  int y = (int) (uv.y() * height);
  x = (x == width) ? x - 1 : x;
  y = (y == height) ? y - 1 : y;
  return color_data[x + y * width];
}
```

1:4 • Name: Yang Hongdi
student number: 2019533234
email: yanghd@shanghaitech.edu.cn

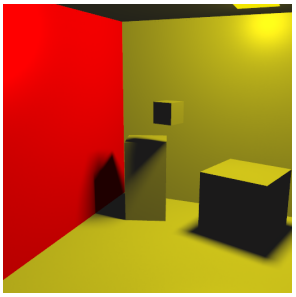
3 RESULTS



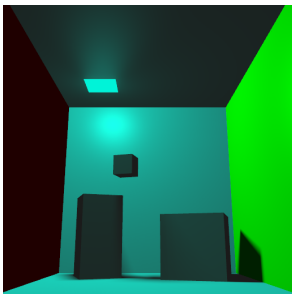
(a) Scene0 with no Msaa



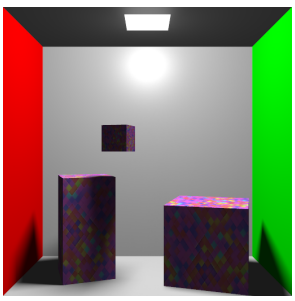
(b) Scene0



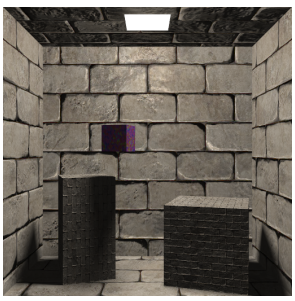
(c) Scene1



(d) Scene2



(e) Scene0 with texture



(f) Scene0 with normal mapping

Fig. 4. Result