Project Proposal for CG100433 course

Ray Tracing Spheres with Variable light

Team member

1750744	韩书琪
1750871	葛超
1751151	郭思远
1752132	王森
1752227	陈喆
1752714	曾佳

Abstract

With the continuous pursuit of the reality of the scene, ray tracing is more and more widely used in 3D scene. Using ray tracing, the authenticity of the scenes is greatly improved, so as to achieve a more perfect visual effect. The amazing effect of ray tracing makes us interested in learning ray tracing.
This project realizes the ray tracing in the 3D scene. The sky box is implemented to construct the scene. We create several models and put them in it and using ray tracing with variable lights. We also create a real-time scene which have three balls. The related technologies include 3D modeling, normal mapping, sky box, ray tracing, etc.

1 Motivation

In view of the great simulation result in scene, ray tracing algorithm is generally considered as the core content of computer graphics, as well as the future direction of game design, movie special effects and other related fields. In recent years, due to the rapid improvement of hardware system, ray tracing algorithms based on distributed GPU and even real-time rendering have emerged. In 3dsmax, Maya and other software, ray tracing algorithm is also used to present the best visual effect on the screen. Compared with the traditional rasterization algorithm, ray tracing can bring more shocking effects. This makes us full of interest and curiosity to learn ray tracing algorithm.

In addition, ray tracing is also very difficult and challenging. By tracing the interaction between ray and object surface, it can get the model of ray path, which is derived from the general technology of geometric optics and has not been widely used. Learning ray tracing is an attempt of the course. We hope to learn some principles and algorithms in depth and complete the whole project with innovation later.

2 The Goal of the project

1. Realize sky box.
2. Realize ball for basic ray tracing with material, texture.
3. Realize other modeling.
4. Perfect the scene to achieve real-time ray tracing effect

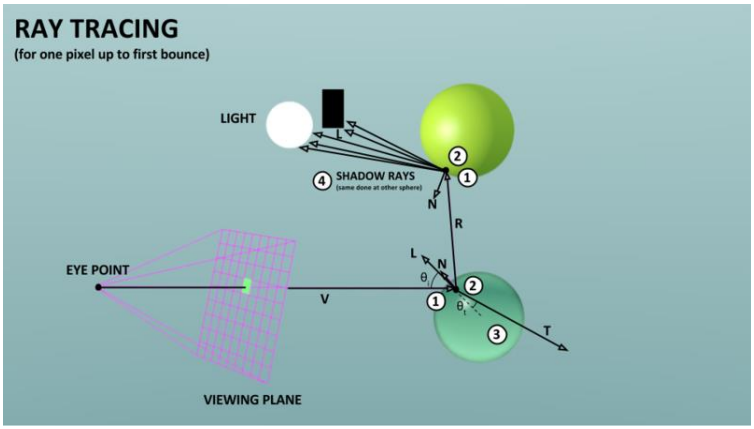
3 The Scope of the project

Build several static object models with material texture and shadow processing.
Build the sky box as the background. The ray tracing algorithm is implemented on the models to map the surrounding environment.
Limited to hardware, skill & time, the dynamic effect & real-time changing background is not so satisfying as we expect.

4 Involved CG techniques

1. Ray Tracing——Recursive ray tracing algorithm

This algorithm was described Turner Whitted in 1979. Previous algorithms traced rays from the eye into the scene until they hit an object, but determined the ray color without recursively tracing more rays. Whitted continued the process. When a ray hits a surface, it can generate up to three new types of rays: reflection, refraction, and shadow. A reflection ray is traced in the mirror-reflection direction. The closest object it intersects is what will be seen in the reflection. Refraction rays traveling through transparent material work similarly, with the addition that a refractive ray could be entering or exiting a material. A shadow ray is traced toward each light. If any opaque object is found between the surface and the light, the surface is in shadow and the light does not illuminate it. This recursive ray tracing added more realism to ray traced images.

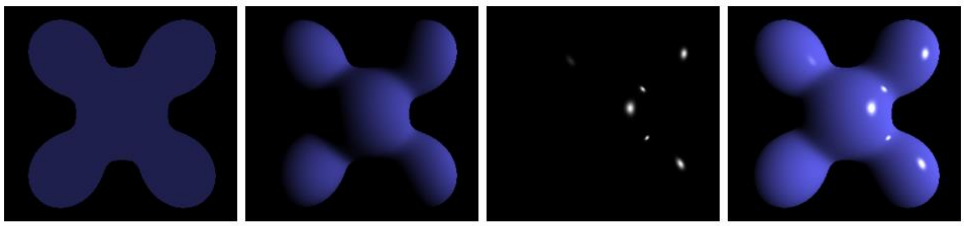


In this project, we first implement a complete space vector class, including intersection detection, calculation of light reflection and refraction. The recursive calculation of the reflected and refracted rays can be used to calculate the original light color. If the light does not

encounter any object, it returns to the background color. In the process of recursion, the light will continuously decay. Therefore, you can limit the depth of the iteration or stop iteration when the light is less than a specific value. Draw the light color corresponding to each pixel to the corresponding coordinates of the picture file, and finally output the drawn image.

2. Phong illumination

Phong reflection is an empirical model of local illumination. It describes the way a surface reflects light as a combination of the diffuse reflection of rough surfaces with the specular reflection of shiny surfaces. It is based on Phong's informal observation that shiny surfaces have small intense specular highlights, while dull surfaces have large highlights that fall off more gradually. The model also includes an ambient term to account for the small amount of light that is scattered about the entire scene.



In this project, we superimpose diffuse reflection and specular reflection, and highlight the specular reflection to make the halo effect more obvious.

3. Antialiasing——Sampling

In order to avoid that the edges of the shadows are too sharp and lack the smoothness of the shadows in reality, we have used a combination of sampling and Monte Carlo algorithms to render our model.

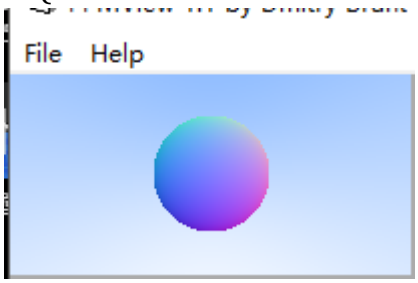
Sampling:

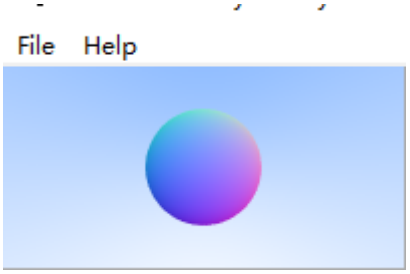
In this project, we have randomly taken several points in the light source sphere, and set the light source at one of them each time to calculate the shadow. This is calculated several times, and each time the light source position is slightly different, so the shadow is also different. These results are superimposed. According to probability knowledge, it is easy to find that when the number of points reached reaches a certain

value, the superimposed shadows are closer to the real.

Method: Sampling total value = Σpixel_value (each coordinate component + one [0,1) surrounding sampling coordinates formed by random value)
Sampling result = total number of samples / number of samples
std :: uniform_real_distribution generates a random value of (0,1) by default
std :: mt19937 is a random generation algorithm, use this algorithm to initialize the one above

We made a ball for test, and then use the sampling method to take the average value of the surrounding 50 random points and compare them(resolution is both 200 * 100):





4. Cubic Mapping: Implementation of scene descriptions such as SkyBox
5. Material: Implementation of material changes of objects such as small balls

5 Project contents

1.We have implemented basic ray tracing ball model with material

2.We have realized scene descriptions such as SkyBox.

3.We have added other models such as planes to achieve more complex ray tracing effects.

4. We have achieved the effect of crystal ball.

5.We have perfected the scene to achieve real-time ray tracing effect

6 Implementation

Referencing 3 books: Ray Tracing in One Weekend, Ray Tracing the Next Week & Ray Tracing the Rest of Your Life, we gradually achieve basic ray tracing system and then add Triangle, Mesh, Skybox, Transform, Model and so on.

Meanwhile, we optimize the management of scene in the form of a tree. The data of scene is generated by traversing the tree.

6.1 Models

6.1.1 Sphere

In graphics, the equation of the sphere in vector form is:

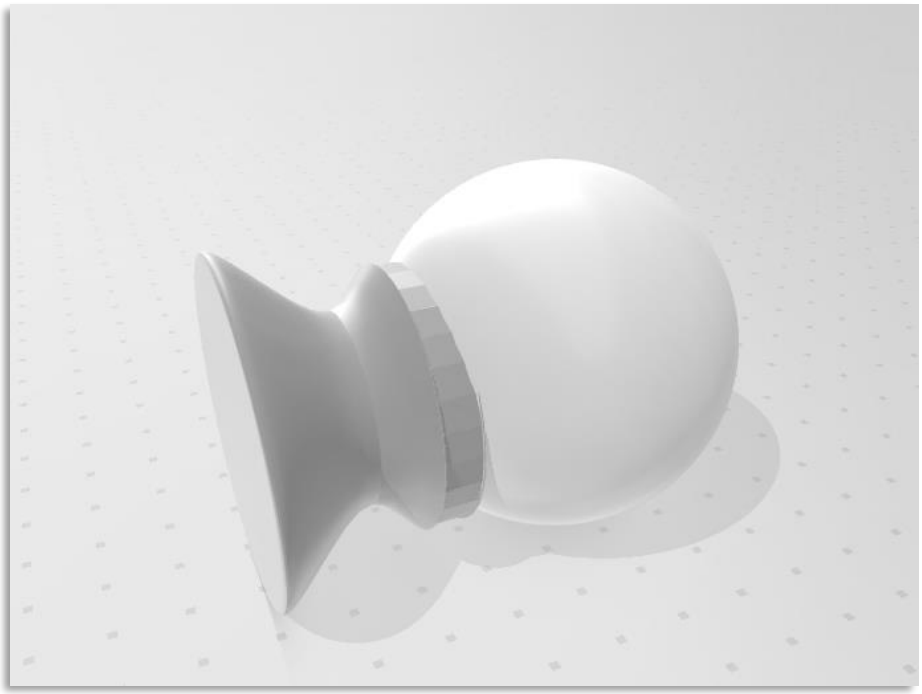
$$dot((p-C),(p-C))=R^2$$

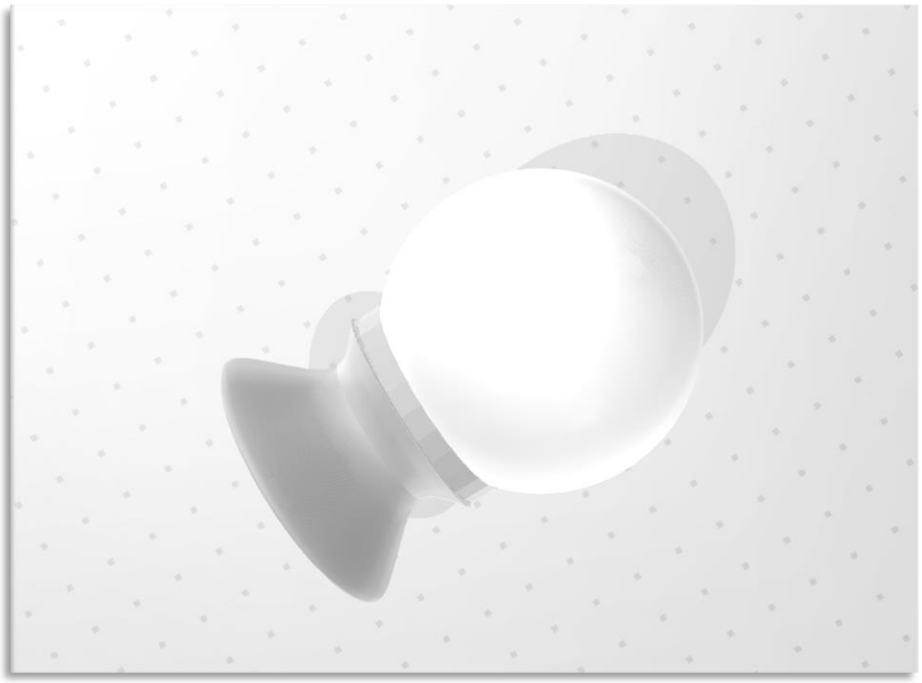
```
struct Sphere{// 6
	float type = 0.0f;
	float matIdx;
	vec3 center;
	float radius;
};
```

6.1.2 Triangle Mesh

In order to achieve highly flexible object, we write the triangle mesh module.

1) using Maya to design a crystal ball model
generate .obj file to import it





2) There are triangles, and then the triangular mesh inherits from 'BVH_Node' to quickly find one of the triangles in a complex model.
A triangle has three vertices, each of which has some properties, such as normal, parametric coordinates, and so on. When a ray intersects with a triangle, it is necessary to calculate the normal direction and parameter coordinates of the intersection.
Ray-Triangle Intersection:

$$\mathbf{e} + t\mathbf{d} = \mathbf{f}(u, v) = \mathbf{a} + \beta(\mathbf{b} - \mathbf{a}) + \gamma(\mathbf{c} - \mathbf{a})$$

$$\beta(\mathbf{a} - \mathbf{b}) + \gamma(\mathbf{a} - \mathbf{c}) + t\mathbf{d} = \mathbf{a} - \mathbf{e}$$

```
struct AABB{// 6
	vec3 minP;
	vec3 maxP;
}

struct Hitable{// 8
	float matIdx;
	float matCoverable;
	struct AABB box;
}

struct Vertex{// 8
	vec3 pos;
	vec3 normal;
	vec2 uv;
}

struct Triangle{// 33
	float type = 3.0;//@0
	struct Hitable base;//@1
	struct Vertex A;//@9
	struct Vertex B;//@17
	struct Vertex C;//@25
};

struct Mesh{
	float type = 4.0;
	struct BVH_Node;
}
```

6.2 Ray Generation

ray equation: p(t)=A+t*B
The function of ray generation:

```
struct Ray{
	vec3 origin;
	vec3 dir;
	vec3 color;
	float tMax;
	float curRayNum;
};
Ray::Ptr GenRay(float s, float t){
```

```
    auto ray = ToPtr(new Ray);
     vec2 rd = lenR * Math::RandInCircle();
    vec3 origin = pos + rd.x * right + rd.y * up;
    vec3 dir = BL_Corner + s * horizontal + t * vertical - origin;
    ray->Init(origin, dir);
}
```

Before encountering the light source, the physical meaning of color is attenuation; After encountering the light source, this value is the color of the light reaching the camera.
'tMax' indicates the farthest distance that light can reach.

Manage data of ray in 3 textures

```
texture0: origin, curRayNum// how many more rays need to be generated inside each pixel
texture1: dir, tMax
texture2: color, time
```

Use 2 buffers:1 for reading, 1 for writing

If our ray hits the sphere, there is some t for which p(t) satisfies the sphere equation:

$$dot((p(t)-C),(p(t)-C))=R^2$$

6.3 Materials

6.3.1 Diffuse Materials

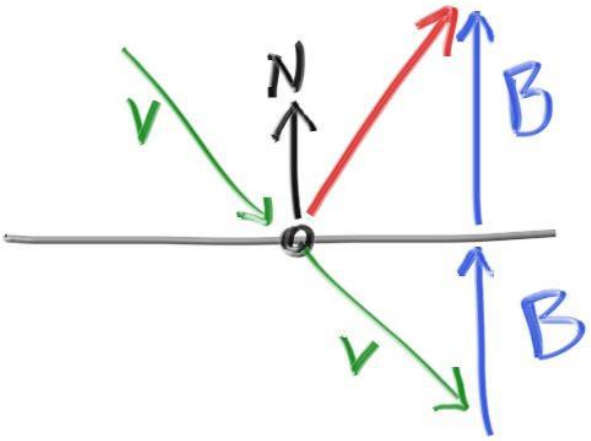
Diffuse objects that don't emit light merely take on the color of their surroundings, but they modulate that with their own intrinsic color. Light that reflects off a diffuse surface has its direction randomized. (approximates mathematically ideal Lambertian)

```
vec3 color(const ray& r, hittable *world) {
    hit_record rec;
    if (world->hit(r, 0.0, MAXFLOAT, rec)) {
        vec3 target = rec.p + rec.normal + random_in_unit_sphere();
        return 0.5 * color(ray(rec.p, target - rec.p), world);
    }
    else {
        vec3 unit_direction = unit_vector(r.direction());
        float t = 0.5*(unit_direction.y() + 1.0);
        return (1.0-t)*vec3(1.0, 1.0, 1.0) + t*vec3(0.5, 0.7, 1.0);
    }
}
class lambertian : public material {
    public:
```

```
        lambertian(const vec3& a) : albedo(a) {}
        virtual bool scatter(const ray& r_in, const hit_record& rec,
                             vec3& attenuation, ray& scattered) const {
            vec3 target = rec.p + rec.normal + random_in_unit_sphere();
            scattered = ray(rec.p, target-rec.p);
            attenuation = albedo;
            return true;
        }

        vec3 albedo;
};
```

```
struct Lambertian{// 4
    float type = 0.0f;
    vec3 color;
};
```

6.3.2 Metal



The reflected ray direction in red is just (v+2B). In our design, N is a unit vector, but v may not be. The length of B should be dot(v,N). Because v points in, we will need a minus sign, yielding:

```
vec3 reflect(const vec3& v, const vec3& n) {
    return v - 2*dot(v,n)*n;
}
class metal : public material {
    public:
        metal(const vec3& a) : albedo(a) {}
        virtual bool scatter(const ray& r_in, const hit_record& rec,
                             vec3& attenuation, ray& scattered) const {
            vec3 reflected = reflect(unit_vector(r_in.direction()), rec.normal);
            scattered = ray(rec.p, reflected);
            attenuation = albedo;
            return (dot(scattered.direction(), rec.normal) > 0);
```

```
        }
        vec3 albedo;
};
```

```
struct Metal{// 5
    float type = 1.0;
    vec3 color;
    float fuzz;
};
```

6.3.3 Dielectrics

Don’t achieve attenuation anymore, because the attenuation Const 'parameter is hard to set, it's always going to be' (0,0,0) ', that's a waste of computing resource.The similar effect of glass color can be simulated by Volume.

```
class dielectric : public material {
    public:
        dielectric(float ri) : ref_idx(ri) {}
        virtual bool scatter(const ray& r_in, const hit_record& rec,
                             vec3& attenuation, ray& scattered) const {
            vec3 outward_normal;
            vec3 reflected = reflect(r_in.direction(), rec.normal);
            float ni_over_nt;
            attenuation = vec3(1.0, 1.0, 0.0);
            vec3 refracted;

            if (dot(r_in.direction(), rec.normal) > 0) {
                outward_normal = -rec.normal;
                ni_over_nt = ref_idx;
            }
            else {
                outward_normal = rec.normal;
                ni_over_nt = 1.0 / ref_idx;
            }

            if (refract(r_in.direction(), outward_normal, ni_over_nt,
refracted)) {
                scattered = ray(rec.p, refracted);
            }
            else {
                scattered = ray(rec.p, reflected);
                return false;
            }

            return true;
        }
```

```
        float ref_idx;
};
```

```
struct Dielectric{// 2
    int type = 2;
    float refractIndex;
};
```

6.4 Scene

Because in CPU we use data structures for this, whereas GLSL has no class, just struct

There are many things to consider, including
AABB, BVH_Node, Group, Hitable, MoveSphere, Sky, Sphere, Transform, Triangle, TriMesh, Volume.

Scene generally speaking is a tree, using Depth First Search. Leaf nodes are Sphere & Triangle. Intermediate nodes are Group, BVH_Node & TriMesh.

6.4.1 Bounding Volume Hierarchies

The ray-object intersection is the main time-bottleneck in a ray tracer, and the time is linear with the number of objects. Because we are sending millions to billions of rays on the same model, we can do an analog of sorting the model and then each ray intersection can be a sublinear search. The two most common families of sorting are to 1) divide the space, and 2) divide the objects. The latter is usually much easier to code up and just as fast to run for most models.
In 3D, boundaries are planes. The equations for the planes are x=x0, and x=x1.
The ray hits the plane x=x0 at the t that satisfies this equation:

$$x_0=A_x+t_0 \cdot B_x$$

t -intervals：

$$t_{x0} = \frac{x_0 - A_x}{Bx}$$

$$t_{x1} = \frac{x_1 - A_x}{Bx}$$

$$t_{x0} = \min(\frac{x_0 - A_x}{Bx}, \frac{x_1 - A_x}{Bx})$$

$$t_{x1} = \max(\frac{x_0 - A_x}{Bx}, \frac{x_1 - A_x}{Bx})$$

The hit function : check whether the box for the node is hit, and if so, check the children and sort out any details:

```
bool bvh_node::hit(const ray& r, float t_min, float t_max, hit_record& rec) const {
```

```
    if (box.hit(r, t_min, t_max)) {
        hit_record left_rec, right_rec;
        bool hit_left = left->hit(r, t_min, t_max, left_rec);
        bool hit_right = right->hit(r, t_min, t_max, right_rec);
        if (hit_left && hit_right) {
            if (left_rec.t < right_rec.t)
                rec = left_rec;
            else
                rec = right_rec;
            return true;
        }
        else if (hit_left) {
            rec = left_rec;
            return true;
        }
        else if (hit_right) {
            rec = right_rec;
            return true;
        }
        else
            return false;
    }
    else return false;
}
```

As long as the list of objects in a bvh_node gets divided into two sub-lists, the hit function will work. At each node split the list along one axis :
(1)randomly choose an axis
(2)sort the primitives using library qsort
(3)put half in each subtree

```
bvh_node::bvh_node(hittable **l, int n, float time0, float time1) {
    int axis = int(3*random_double());

    if (axis == 0)
        qsort(l, n, sizeof(hittable *), box_x_compare);
    else if (axis == 1)
        qsort(l, n, sizeof(hittable *), box_y_compare);
    else
        qsort(l, n, sizeof(hittable *), box_z_compare);

    if (n == 1) {
        left = right = l[0];
    }
```

```
        else if (n == 2) {
            left = l[0];
            right = l[1];
        }
        else {
            left = new bvh_node(l, n/2, time0, time1);
            right = new bvh_node(l + n/2, n - n/2, time0, time1);
        }

        aabb box_left, box_right;

        if (!left->bounding_box(time0, time1, box_left) ||
            !right->bounding_box(time0, time1, box_right)) {

            std::cerr << "no bounding box in bvh_node constructor\n";
        }

        box = surrounding_box(box_left, box_right);
}
```

```
struct BVH_Node{// 10 + (left != NULL) + (right != NULL)
        float type = 2.0;//@0
        struct Hitable base;//@1
       float leftIdx;//if left != NULL
        float rightIdx;//if right != NULL
        float childrenEnd = -1.0;
}
```

Although each node has a bounding box in the data of scene, only the BVH_Node's bounding box is actually used for computation within the shader.
The enclosing boxes of other nodes are useless, so their enclosing boxes are no longer generated

Shader

```
struct HitRst RayIn_Scene(){
        Stack_Push(9);// location of Group's child's pointer
        struct HitRst finalHitRst = HitRst_InValid;
        while(!Stack_Empty()){
            float pIdx = Stack_Top();
          float idx = At(SceneData, pIdx);
          if(idx == -1.0){
              Stack_Pop();
              if(Stack_Empty())
                  return finalHitRst;
```

```
            Stack_Acc();
            continue;
        }

        float type = At(SceneData, idx);
        if(type == HT_Sphere){
            struct HitRst hitRst = RayIn_Sphere(idx);
            if(hitRst.hit)
                finalHitRst = hitRst;

            Stack_Acc();
        }
        else if(type == HT_Group)
            Stack_Push(idx+9);
        else if(type == HT_BVH_Node){
            if(AABB_Hit(idx+3))
                Stack_Push(idx+9);
            else
                Stack_Acc();
        }
    }

    return finalHitRst;
}
```

6.4.2 skybox

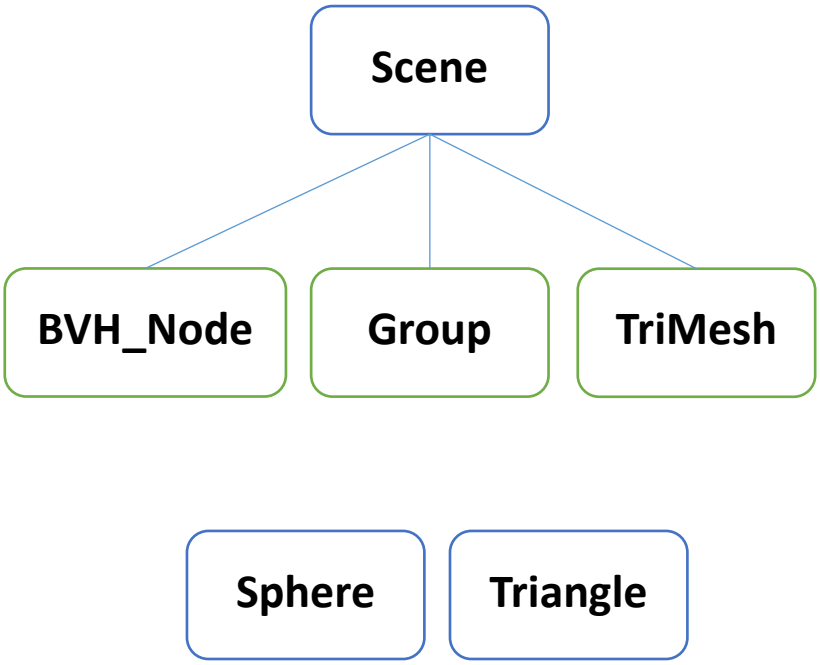
loads a cubemap texture from 6 individual texture faces

```
Skybox::Skybox(const vector<string> & skybox) {
    if (skybox.size() != 6)
        return;

    // order:
    // +X (right)
    // -X (left)
    // +Y (top)
    // -Y (bottom)
    // +Z (front)
    // -Z (back)
    // -------------------------------------------------------
    for (size_t i = 0; i < 6; i++)
    {
        auto img = new Image(skybox[i].c_str(), false);
        if (!img->IsValid()) {
```

```
                printf("ERROR: Skybox texture failed to load at path: %s\n",
skybox[i].c_str());
                imgs.clear();
                return;
            }

            imgs.push_back(CppUtil::Basic::CPtr<Image>(img));
        }
}
```

6.4.3 Generation of Data of scene: Problem of RayIn Recursion

GLSL does not allow functions to recurse. Therefore, "recursive expansion" technique is needed. The scene, in general, is a node tree. The leaf nodes (Sphere and Triangle) are drawable and the intermediate nodes are Group, BVH_Node and TriMesh. The search process is a DFS. So Recursive expansion is actually a expansion on a multiway tree using stack.



GLSL has no Pointers. But the principle of a pointer is an address, which is equivalent to an array index.

Non-recursive expansion process:
Loop {
Out of the stack
Process the node
Pushes all children of the node
}

The space complexity of such a stack is' O(bd) ', where b is the width and d is the depth.

```
struct HitRst RayIn_Scene(){
	int idxStack[100];
	int idxStackSize = 0;
	struct HitRst finalHitRst = HitRst_InValid;

	idxStack[idxStackSize++] = 0;
	while(idxStackSize > 0){
		int idx = idxStack[--idxStackSize];

		int type = int(SceneData[idx]);
		if(type == HT_Sphere){
			struct HitRst hitRst = RayIn_Sphere(idx);
			if(hitRst.hit)
				finalHitRst = hitRst;
		}
		else if(type == HT_Group){
			int childrenNum = int(SceneData[idx+3]);
			for(int i=0; i < childrenNum; i++)
				idxStack[idxStackSize++] = int(SceneData[idx+4+i]);
		}
		//else
		//	;// do nothing
	}

	return finalHitRst;
}
```

6.4.4 Transport of data of scene

There are 4 ways to get data in fragment shader:
array of const, Uniform, in, Texture
Since the data is not vertex data, but more complex scenario data, we exclude in
Const arrays &uniform array are also indicated a lack of space, because the data of scene is too large.
The previous implementation is to put the data in the const global variable, and as the scene gets larger, the data grows more. A const global variable will cause an error, indicating insufficient space .It's the same in Uniform.
So now we can only put it in the texture. Think of one-dimensional textures as arrays. Since 'Sampler1D' has been removed from the current version of OpenGL, we can use a two-dimensional texture with height == 1.

We no longer need to dynamically generate Fragment Shader. We just need to prepare the data.
After a lot of trials and errors, the most reasonable square texture was finally adopted to store the data.
The data is divided into two kinds:
1) one kind is to describe a organization of scene. Just need to take a float. So use a single channel of texture to store.
2) The other is a large pieces of data, such as matrix, the vertex data, etc. Put it in the four-channel texture, which reduces the texture sampling frequency.

The center and radius of the ball are combined into a pack4.
BVH_Node's enclosing box is placed in 2 pack4 (minP, 0), (maxP, 0).
Put Vertex in 2 pack4 (pos,u), (normal, v)
Put Mat4 in 4 pack4, and Mat3 in 3 pack4

```
float At(sampler2D data, float idx){
      float row = (idx+0.5)/textureSize(data, 0).x;
      float y = (int(row)+0.5)/textureSize(data, 0).y;
      float x = (idx + 0.5 - int(row) * textureSize(data, 0).x)/textureSize(data, 0).x;
       vec2 texCoords = vec2(x, y);
       return texture2D(data, texCoords).x;
}

void GetPack(float idx, out vec4 pack){
      float row = (idx+0.5)/textureSize(PackData, 0).x;
      float y = (int(row)+0.5)/textureSize(PackData, 0).y;
      float x = (idx + 0.5 - int(row) * textureSize(PackData,
0).x)/textureSize(PackData, 0).x;
       vec2 texCoords = vec2(x, y);
      pack = texture2D(PackData, texCoords);
}
```

```
struct Sphere{
       float type = 0.0;
       float matIdx;
       float isMatCoverable;
       float center_radius_pack4_idx;
}

struct Group{
       float type = 1.0;
       float matIdx;
       float isMatCoverable;
       float children[len];
       float childrenEnd;
```

```
}

struct BVH_Node{
      float type = 2.0;
       float matIdx;
       float isMatCoverable;
       float AABB_pack3_idx;
       float leftIdx;//if left != NULL
       float rightIdx;//if right != NULL
       float childrenEnd = -1.0;
}

struct Triangle{
       float type = 3.0;
       float matIdx;
       float isMatCoverable;
       float vertexABC_pack4_idx;
}

struct TriMesh{
       float type = 4.0;
       float matIdx;
       float isMatCoverable;
       float AABB_pack4_idx;
       float leftIdx;//如果 left != NULL，则有此域
       float rightIdx;//如果 right != NULL，则有此域
       float childrenEnd = -1.0;
}

struct Transform{
       float type = 4.0;
       float matIdx;
       float isMatCoverable;
       float tfm_pack4_idx;
       float boundary;//如果 boundary != NULL，则有此域
       float childEnd = -1.0;
}
```

6.4.5 Optimization of scene's structure

When we use stack, there is a bug and we don't know why. We can only use 20-length

arrays , and we need to initialize them.

If we can only use 20-length stacks , the stack is insufficient when all the child nodes are loaded directly into the stack when accessing the Group.

So to optimize, we consider a tree structure that allows for a non-recursive depth search with a stack space only of O(depth), whereas the original structure required O(b*depth).

Place the child node pointers in sequence, and -1 represents the end of the child list.

Because $2^{20} = 1048576$, now there is no problem with general large models and large scenes. Unless the larger model is put in a too deep position.

After testing, the stack space can be increased again, so the last problem of the structure was solved successfully.

Hitable structure

```
struct Sphere{// 7
	int type = 0;
	float matIdx;
	bool matCoverable;
	vec3 center;// @3
	float radius;
};

struct Group{// 4 + childrenNum
	int type = 1;
	float matIdx;
	bool matCoverable;
	int childrenIdx[childrenNum];
	int childrenEnd = -1;
}
```

Process:

1.Push to get the pointer of the current node pointer, and look up the table to get the node pointer

2. If the current node is a leaf node (Sphere, Triangle), then executive

Acc(): Push(Pop()+1)

after processing

3. If the current node is a branch node (Group, BVH_Node, Mesh), push the pointer of its child node pointer to the stack

4. If the current node is an empty node (-1), then

Pop(); CheckEmpty (); Exit (table [Top ()]), Acc ()

6.4.6 example

A Group node has two child nodes, both of which are Sphere.The program will add one to that node and put it in a Group.

SceneData:

```
/*@   0*/ 1, -1, 1,   5, -1
/*@   5*/ 1, -1, 1, 11, 18, -1,
/*@ 11*/ 0,   2, 0,   0, 0, -1, 0.5,
/*@ 18*/ 0,   0, 0,   0, 0, -2, 0.5
```

Process:

1. Push 3 // 初始化, Group 的第一个孩子节点指针的位置 //[3 >
2. Top-->3, table[3]-->5, Group,
   Push 8 == 5+3; //[3, 8 >
3. Top-->8, table[8]-->11, Sphere,
   Acc 8-->9 //[3, 9 >
4. Top-->9, table[9]-->18, Sphere,
   Acc 9-->10 //[3, 10 >
5. Top-->10, table[10]-->-1, NULL,
   Pop, Not Empty, Exit(table[Top-->3]-->5), Acc 3-->4 //[4 >
6. Top-->4, table[4]-->-1, NULL,
   Pop, Empty, return //[>

The new structure is quite reasonable. There is no need to expand the scene (just to keep the tree) and no need to add space (just change 'childrenNum' to 'childrenEnd'). Moreover, it can fix the problems that need to be handled when leaving the parent node (such as setting 'Material' for branch node, 'Transform' for ray).

6.4.7 stack optimization

Because it is found that stack size greatly affects speed, it is important to optimize the push variables.

The most frequent pushed is the pointer of the parent node pointer and the pointer of the child node pointer.

The latter is necessary, and the former is to handle the tail operation of the group node.

We could use the pointer A of the parent node pointer as the tail of the child node array. It was originally -1, but now it's -A, to save one push operation and to get A directly.

6.6 Transform

The Transform node has a child node that is treated like a Group in the data structure.

```
struct AABB{// 6
    vec3 minP;
```

```
	vec3 maxP;
}

struct Hitable{// 8
	float matIdx;
	float matCoverable;
	struct AABB box;
}

struct Transform{// 52
	float type = 5.0;//@0
	struct Hitable hitable;//@1
	mat4 transform;//@9
	mat4 invTransform;//@25
	mat3 normTransform;//@41
	float childIdx;//@50
	float childEnd = -1.0;//@51
}
```

Operations in and out of nodes:

Actually, when executing, we need to know that a collision occurred inside the transform.

The same is true for Group, BVH_Node and TriMesh.

We can record the value of tMax at the time of entry. If the value of tMax is smaller than that at the time of entry, a collision has occurred.

We can put tMax in the stack and push the pointer of the child pointer before it is pushed.

6.6 Solid Textures

A texture in graphics means a function that makes the colors on a surface procedural.

```
class texture (
	public:
		virtual vec3 value(float u, float v, const vec3& p) const = O;
};

class constant_texture : public texture {
	public:
		constant_texture() {}
		constant_texture(vec3 c) : color(c) {}
		virtual vec3 value(float u, float v, const vec3& p) const {
			return color;
		}
```

```
        vec3 color;
};
```

Image Texture Mapping

Use image utility stb_image.

```
struct ConstTexture{// 4
    float type = 0.0;
    vec3 color;
}

struct ImgTexture{// 2
    float type = 1.0;
    float texArrIdx;//Uniform TexArr[MAX_TEXTURE_SIZE]
}
```

6.7 Light

Get color from Texture, related to intersection angle & distance

```
bool Light::Scatter(const HitRecord & rec) const {
    float d = rec.ray->GetTMax() * length(rec.ray->GetDir());
    float attDis = 1.0f / (1.0f + d * (linear + quadratic * d));
    float attAngle = abs(dot(normalize(rec.ray->GetDir()), rec.vertex.normal));
    rec.ray->SetLightColor(attDis * attAngle * lightTex->Value(rec.vertex.u,
rec.vertex.v, rec.vertex.pos));
    return false;
}
```

6.8 Volumes

Things like smoke/fog/mist are sometimes called volumes or participating media. Subsurface scattering is sort of like dense fog inside an object. A bunch of smoke can be replaced with a surface that probabilistically might or might not be there at every point in the volume.

As the ray passes through the volume, it may scatter at any point. The denser the volume, the more likely that is. The probability that the ray scatters in any small distance δL is:

$$\text{probability} = C \bullet \delta L$$

where C is proportional to the optical density of the volume.

Handling Volume is troublesome because it needs to generate new light to perform auxiliary calculations.

A new ray is generated, and if we want to trace it, we can only push the original ray against the original result of collision.
Then push the boundary, and then do the follow-up operations when it finally returns to Volume
Needs to emit light three times to calculate the walking length of light in the Volume.
1) The first time
If boundary is empty, it ends
Stack the original ray and the original collision result
To press the tMax
push state 1 here to show that this is the first time
Stack the pointer of the Volume pointer and the boundary pointer
Use the original ray to calculate the boundary collision
2) The second time
When back to Volume,pop state 1.Do the second processing
Pop tMax out, and if there is no collision, restore the light and the collision result, End
Else enter the second processing:
Push boundaryHitRst, state 2 and pointer of Volume pointer to the stack. Push boundary.
Use the original ray to initialize the reverse ray and initialize the collision result
Do ray calculations
3) The third time
When you return to Volume, pop state 2.
the third processing :
If a collision occurs in the reverse ray, pop the original collision result. The Volume can be calculated directly according to the formula
Else
Initializes the inner ray with a reverse ray
Push status 3, push the pointer of the Volume pointer, push the boundary
4) end
When returing to Volume, pop state 3.
Pop tMax.
If no collision occurs, restore the light and the collision result. End.
If a collision occurs, the Volume is calculated according to the formula.

Hitable

```
struct AABB{// 6
	vec3 minP;
	vec3 maxP;
}

struct Hitable{// 8
	float matIdx;
	float matCoverable;
	struct AABB box;
```

```
}

struct Volume{
	float type = 6.0;//@0
	struct Hitable hitable;//@1
	float density;//@9
	float boundaryIdx;//@10
	float childEnd = -1.0;//@11
}
```

6.9 Random number

```
float RandXY(float x, float y);
float Rand(vec2 TexCoords, float frameTime, int rdCnt);
vec2 RandInSquare();
vec3 RandInCube();
vec2 RandInCircle();
vec3 RandInSphere();
```

6.10 Camera

```
struct Camera{
	vec3 pos;
	vec3 BL_Corner;
	vec3 horizontal;
	vec3 vertical;
	vec3 right;
	vec3 up;
	vec3 front;
	float lenR;
	float t0;
	float t1;
};
```

7 Optimization

7.1 AABB

Although in the scene data, each node has a bounding box, in fact, only the BVH_Node's bounding box is used for calculation in the shader. The bounding boxes of other nodes are useless, so their bounding boxes are no longer generated.

7.2push tMax into the stack

Only those group nodes that have material and nodes that need collision information (Transform, Volume) need to push tMax into the stack.

7.3 Put data in three- and four-channel packs

Large blocks of data, such as matrices and vertex data, we put it in a four-channel texture, which can reduce the number of texture samples.

7.4 optimization of pushing into stack

Because it is found that the size of the stack greatly affects the speed, it is very important to optimize the variables on the stack

The most frequent pushes to the stack are pointers to the parent node pointers and child node pointers.

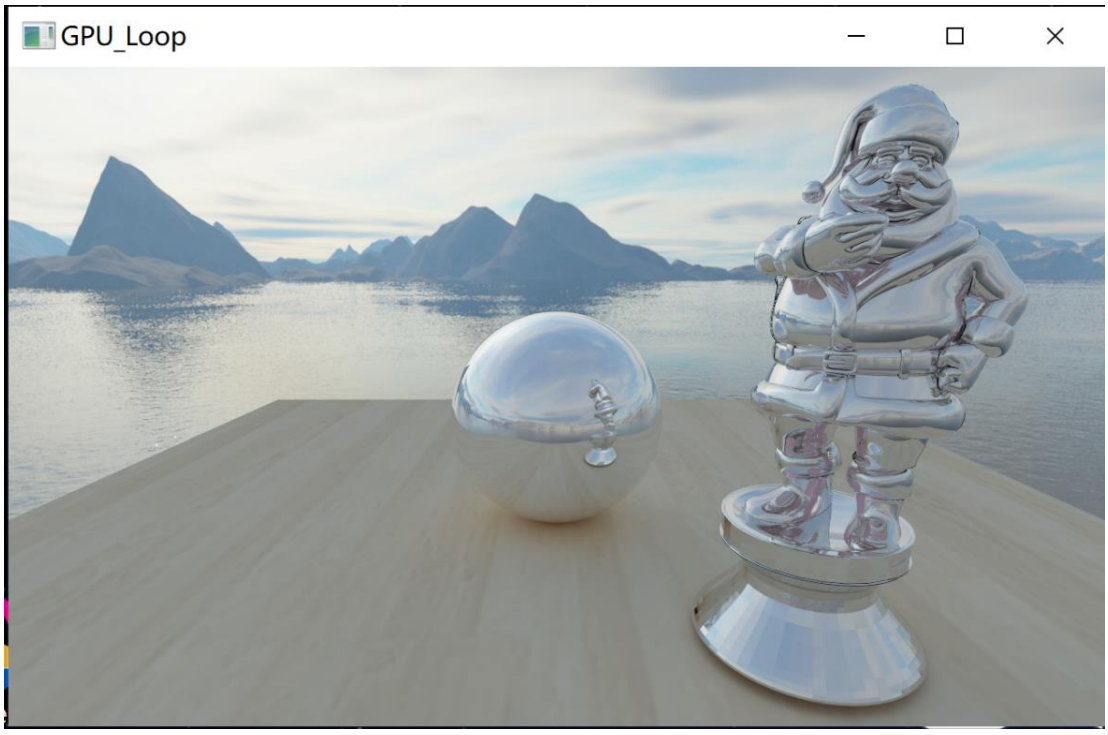
The latter is necessary, the former is to deal with tail operations of group nodes
We use the pointer A of the parent node pointer as the tail of the child node array. It was originally -1, but now it is changed to -A. One time can save one time on the stack, and the other time you can directly get A.

8 Results

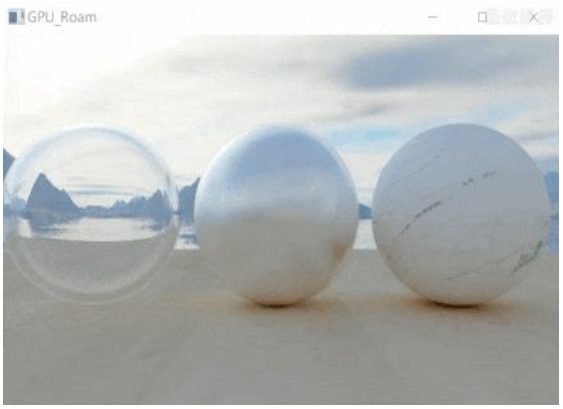
Parameter:

```
bin option = 111100;

# 根目录
string RootPath = "${CMAKE_SOURCE_DIR}";

# 默认输出图像大小
int val_Default_ImgWidth = 1280;
int val_Default_ImgHeight = 768;

# 图像更新帧率
int FPS = 1;

# 采样率
int sampleNum = 2;

# 高斯模糊次数
int blurNum = 1;
```

Ray Tracing-static:



Ray Tracing-Real-time：(Double click to play)



9 Summary & Advantages

Although one of our models is static, we are the only group that implements ray tracing on aspheric models.

Aspheric models are made up of thousands of triangles, how to handle and store them is troublesome. We have designed a tree structure in order to speed up the fragment shader. The complexity of the stack space is O (depth) when searching at non-recursive depths, and it doesn't need to expand the scene or add space. It can also handle situations where operations need to be handled when leaving the parent node.

We also used BVH_Node to speed up the scene traversal. After testing, we have increased the speed by about 7 times after using BVH, and the effect is significant.

We encountered many difficulties during the time of completing the project, and we did our best to try to solve them. For example, when writing a fragment shader, we used a const array in the early days, but the scene data was slightly large, and the compilation reported an error, indicating that the space was insufficient. We then switched to uniform arrays, which again indicated a lack of space. So we had to use texture.

Because all we want is an array that can hold data, so we used a texture with width as the data length and height as 1 to store the scene data. But the scene is bigger, when the array length reaches hundreds of thousands, an error occurred again. OpenGL does not support widths as large as hundreds of thousands.
After many trial and error, we finally adopted the most reasonable square texture to store the data.

10 Roles in group

1750744	韩书琪	Skybox& Report
1750871	葛超	Shader& Scene& Debug& Optimize
1751151	郭思远	Basic Frame & Report
1752132	王森	Raytracing System& Model & Debug &Optimize
1752227	陈喆	Model & Optimize
1752714	曾佳	Texture & Report & Presentation

10 References

[1] Ray Tracing in One Weekend
https://raytracing.github.io/books/RayTracingInOneWeekend.html
[2] 赫恩（Hearn, D.），巴克（Baker, M. P.），卡里瑟斯（Carithers, W. R.）著，蔡士杰，杨若瑜译. 计算机图形学[M]. 4 版. 北京：电子工业出版社，2014.
[3] 施莱尔（Shreiner, D.）等著，王锐等译. OpenGL 编程指南[M]. 8 版. 北京：机械工业出版社，2014.
[4] learnopengl https://learnopengl.com/Getting-started/OpenGL