Project 5: Whatever

Name: Minsuk Kim(m.kim)
Instructor: Dr. Gary Herron
Class: CS500
Semester: Spring 2022
Date: 4/20/2022

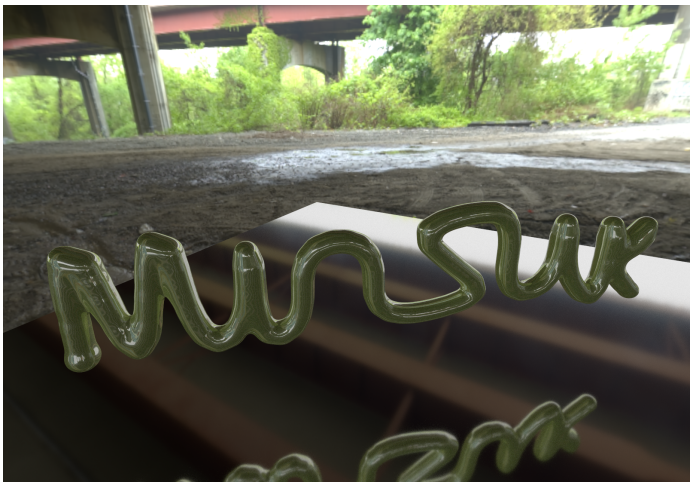


Table of Contents

Introduction

Overview

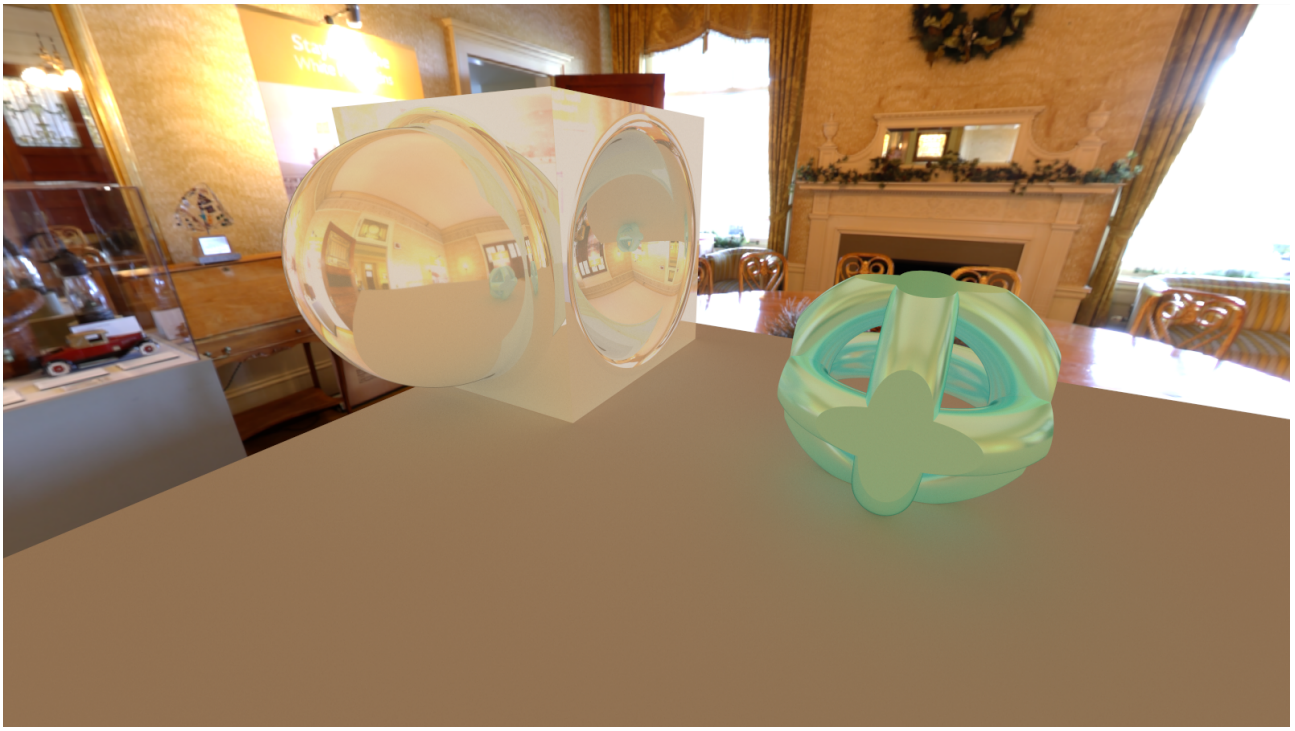
The main purpose of the project is to implement image-based lighting and constructive solid geometry with signed distance field tracing. The program will calculate the lighting value on all incoming lighting from the environment map. The program calculates the signed distance between the object and ray to trace the constructive solid geometry. I started the project with the previous project that was about BRDF. I also used an external library Assimp that read the object files and BVH for bounding volume hierarchy.

Implementation

First, the program loaded the scene file and created the shapes for each object file. The program casts rays for each pixel. When the ray collides with a certain object that is not a light object, the ray will generate another ray that towards a random direction. The random direction is a distributed vector around a normal vector in the hit position. The program also uses the reflection feature that random direction is a distributed vector around a reflective vector. The program produces the randomly distributed refraction vector as well. Since we used the environment map for the incoming lights surrounded by the whole scene, the rays that did not hit the object will calculate the incoming lights from the environment map. The program uses a signed distance estimate for testing the intersection with a complex geometry object. The program supports the union, subtraction, and intersection between those complex geometry objects. The program repeatedly generates rays to the same position for better accurate light calculation. The program also uses anti-aliasing by changing random offset to a pixel position in each pass.

Result Images

sample image



The image shows two complex geometry. The left object is a box subtracted by one sphere and added(union) by one sphere. The right object is the box that intersects with three torus(aligned X axis, Y axis, Z axis). The subtraction and union CSG operation work smoothly at the boundary. All objects' light calculations work with the environment map.

The image shows the curve made from complex geometry made up of spheres. The positions of the spheres are calculated with the bezier curve using the De Boor algorithm. Each sphere is added to the other smoothly with the CSG operation(union). The curve shows the word ‘Minsuk’ with the specific control points I put. The curve is affected by the environment map lighting.

Notes

Image Based Lighting

To compute all incident radiance came from the environment map, I pre-process the environment map with marginal and conditional cumulative density function for computing the probability for image based light. The conditional cumulative density function will be calculated by

$$P(v|u) \approx \sum_{i=1}^{v} p(u,i)$$

,where $p(u,v) \approx L(u,v) \cdot sin\theta$

, where $L(u,v) = 0.2126R(u,v) + 0.7152G(u,v) + 0.0722B(u,v)$

and the margianl cumulative density function will be calculated by

$$P(u) \approx \sum_{i=1}^{u} P(h|i)$$

Then, when calcaculate the point on the skysphere(mapped with the environment map), I use the cumulative density function. I find the position uv on the texture coordinate by calculating :

$$u = DU(P(W) * \xi_1)$$

$$v = DV(P(H|u) * \xi_2)$$

,where ξ_1 and ξ_2 are random number and W is width of the texture and H is height of the texture.

DU(X) is the index on array of marginal cumulative density function that has maximum value that is not larger than X

DV(X) is the index on array of conditional cumulative density function that has maximum value that is not larger than X

After calculate the u and v, I convert the u and v into three-dimensional coordinate to find the position and normal on the skysphere.

To find the pdf of light,

$$pdfU = 2 * \pi * P(u)/P(W)$$

$$pdfV = 2 * \pi * P(v|u)/P(H|u)$$

$$pdf = pdfU * pdfV * sin\theta \,/\, (area\ of\ light)$$

, where area of light is r * r * 4 * PI in this project and u and v is texture coordinate converts from the intersection point on skysphere.

To get the radiance color value for the environment map lighting, we just need to convert the three-dimensional point on the skysphere into two-dimensional and get the color value from the texture.

It is implemented on line 586 in raytrace.cpp

CSG

To achieve the complex geometry object for my project, I used ray marching for those objects with the signed distance estimate. To get the distance between the point and simple objects,

Sphere

$$distance(P) = length(P - C) - r$$

, where C is center of sphere and r is radius of sphere

Torus

$$distance(P) = length(v(P)) - r$$
$$v(P) = (length(P.x, P.y) - R, P.z)$$

, where R and r is outer radius and inner radius of torus

Box

$$distance(P) = max(xmax, ymax, zmax)$$

, where xmax, ymax, zmax is max(min - P, P - max)

Plane

$$distance(P) = (N \cdot P) + d$$

, where dot(N, (x, y, z)) + d = 0 is equation for plane

Cone

$$distance(P) = length(P.x, P.y) * cos\theta - abs(P.z) * sin\theta$$

, where theta is angle of cone

Capsule

$$distance(P) = length(P') - r$$
$$P' = P - vec3(0, 0, D)$$
$$D = 0.0 \; if \; P.z < 0 \; and \; H \; if \; P.z > H$$

, where H is height of capsule and r is radius of capsule

To achieve the complex geometry, I made a system for combining two geometry to generate new shapes. The combining object has union, subtraction, intersection system that can be calculated by following:

$$UNION(P, Q) = min(distance(P), distance(Q))$$
$$SUBTRACTION(P, Q) = max(distance(P), -distance(Q))$$
$$INTERSECTION(P, Q) = max(distance(P), distance(Q))$$

To get a better visualization, I made the functionality for smoothly combining two object with following:

$$SMOOTHUNION(P, Q) = lerp(p, q, h) - k * h * (1.0 - h)$$
$$h = 0.5 - 0.5 * (p - q)/k$$
$$SMOOTHSUBTRACTION(P, Q) = lerp(p, -q, h) + k * h * (1.0 - h)$$
$$h = 0.5 - 0.5 * (p + q)/k$$
$$SMOOTHINTERSECTION(P, Q) = lerp(p, q, h) + k * h * (1.0 - h)$$
$$h = 0.5 - 0.5 * (p - q)/k$$

,where k is smooth constant(use 0.25 in this project) and p and q is distance(P), distance(Q)
It is implemented on line 1051 in raytrace.cpp

Curve

For the second image, I made a b-spline curve with the De Boor algorithm and CSG operation. I calculate each position of the curve at a certain distance and place the spheres in the positions. After placing the sphere, I made a smooth union CSG operation to connect them together to make a curve. The points on the curve are calculated with the following:

$$r(t) = \sum_{i=0}^{27} P_i B_i^2(t)$$

, where B is B-spline function and t is [2,28) and knot sequences are {0, …. 30}