Real Time Path Tracing with Implicit Geometries Defined by Distance Functions

Zhuo Lu* Diyu Luo Jiejun Luo

ABSTRACT

Real-time rendering with global illumination can be computationally expensive if a large scene with great details is stored in explicit vertices and edges. In this paper, we attempt to realize the idea of real-time path tracing by drawing two triangles that cover the whole screen and perform sphere path tracing in fragment shader. We model a complex 3-D scene of a castle using signed distance functions, which are more computationally efficient compared to explicit vertices and edges. We then perform ray-marching algorithm to find the intersection between the ray and the scene. The final rendering result is real-time and entails advantages from global illumination, including soft shadows and more realistic visual effects from physically based materials.

CCS CONCEPTS

• **Computing methodologies → Computer graphics**;

KEYWORDS

Ray Marching; Signed Distance Function; Procedural Modeling

ACM Reference Format:
Zhuo Lu, Diyu Luo, and Jiejun Luo. 2018. Real Time Path Tracing with Implicit Geometries Defined by Distance Functions.

1 INTRODUCTION

Procedural modeling is the modeling of scenes using algorithms instead of explicit lists of geometry specified vertex by vertex[3]. There are several reasons to choose procedural modeling. If the scene is too repetitive, it would be computationally expensive to store information about every vertex as it is impossible to store such level of detail — the memory needed would be prohibitively large. It would be desirable if we could model scenes implicitly, but there are hardly mature tools or frameworks available. In this paper, we model a 3-D scene with implicit surfaces like boxes, spheres, cones and cubes and apply transformations like rotation, translation and reflection. Constructive solid geometry is used to take the union, intersection and difference of different objects. We use ray marching algorithm to find the ray intersection on surfaces of objects in the scene. We use three types of BSDF — diffuse, mirror and glass — to model different materials, and bilateral filtering for denoising.

2 METHODS

2.1 Implicit surfaces

Given a function $f(p) : R^n \to R$, we can define a set $\{p | f(p) = c; c \in R\}$, called a "level set", the set of all points where a function $f(p)$ takes value c. For $n = 3$, this set is called an "isosurface" or "implicit surface", as it is the surface implied by the function $f(p)$. For $n = 3$, this set is called an "isosurface" or "implicit surface", as it is the surface implied by the function $f(p)$[3]. In this paper, we combine analytic functions which implies geometric shapes or details to create scenes.

2.2 Signed distance functions

Distance functions are a special kind of function from R^3 to R. $\forall p \in R^3$, a distance function $d(p)$ gives the distance to the closest point of an implicit surface defined by $d(p) = 0$. A signed distance function gives a signed distance to that surface, usually with the distances being negative on the inside of objects[3].

Centered at the origin, the following primitive 3-D objects could be modeled mathematically using singed distance functions.

2.2.1 *Sphere.* Sphere is the simplest object to model using distance functions. The surface of a sphere contains all points that have distance r from the origin:

$$d_{\text{shpere}}(p, r) = |p| - r \tag{1}$$

2.2.2 *Torus.* The signed distance function for torus is

$$d_{\text{torus}}(p, r_1, r_2) = \sqrt{(\sqrt{p_x^2 + p_y^2})^2 - (r_1)^2} - r_2 \tag{2}$$

2.2.3 *Cylinder.* The signed distance function for cylinder is

$$d_{\text{cylinder}}(p, r, h) = max(\sqrt{p_x^2 + p_z^2} - r, |p_y| - \frac{h}{2}) \tag{3}$$

2.2.4 *Cone.* The signed distance function for cone is

$$\begin{gathered} d_{\text{cone}}(p, r, h) = max(\sqrt{p_x^2 + p_z^2}\cos(\theta) - |p_y|\sin(\theta), p_y - h, -p_y) \\ \theta = \arctan(\frac{r}{h}) \end{gathered} \tag{4}$$

2.2.5 *Box.* The signed distance function for box with side length $s = (s_x, s_y, s_z)$ is:

$$d_{\text{box}} = max(|p_x| - \frac{s_x}{2}, |p_y| - \frac{s_y}{2}, |p_z| - \frac{s_z}{2}) \tag{5}$$

2.3 Ray marching

We use ray marching, also known as sphere marching method, to find the implicit surface associated with a signed distance field. This is equivalent to the problem of finding the root of a function $d(p)$ along the ray. Sphere marching makes use of the fact that with a distance function, the minimum distance to the surface at any point in space is known. This gives a minimum safe step distance to the surface, allowing for fast iteration along the ray without missing any surface detail[3]. The trick is to be able to compute or estimate

*Names in alphabetical order.

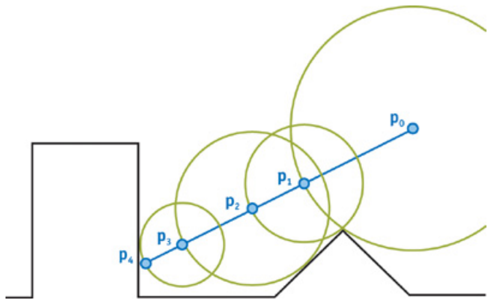


Figure 1: Sphere tracing

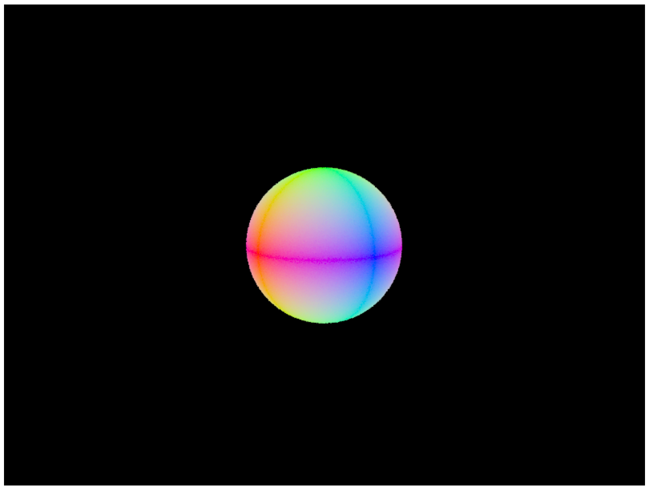


Figure 2: Surface normal approximation of sphere

(a lower bound of) the distance to the closest surface at any point in space, which allows for marching in large steps along the ray[4].

The algorithm proceeds as follows: Similar to ray tracing, we put the camera at a position and place a pixel grid in front of it. For each pixel in the grid, we shoot a ray from the camera through it, and find its intersection with the scene[5]. Since we have an implicit signed distance function, we can not find the root analytically by setting the equation to zero. The basic idea of ray marching is to move along the ray step by step, and carefully check if the current step hits the scene. The naive way would be to take constant step size, however we could speed it up by taking the advantage that we know the distance between the current point and the implicit surface. This provides a safe lower bound of the distance between the current point along the ray and the implicit surface. This method will be significantly faster than constant-size stepping. Sphere tracing significantly improves the speed and accuracy if we take a maximum safe step at each iteration. Figure 4 indicates 4 steps of marching along the ray from origin p_0.

Ray marching inside an object turns out to the same problem but with a twist. Since signed distance function is defined as the distance to the closest surface, to find the point that the ray exits the object, we can simply flip the sign of the signed distance function and apply the same algorithm to find the closest intersection.

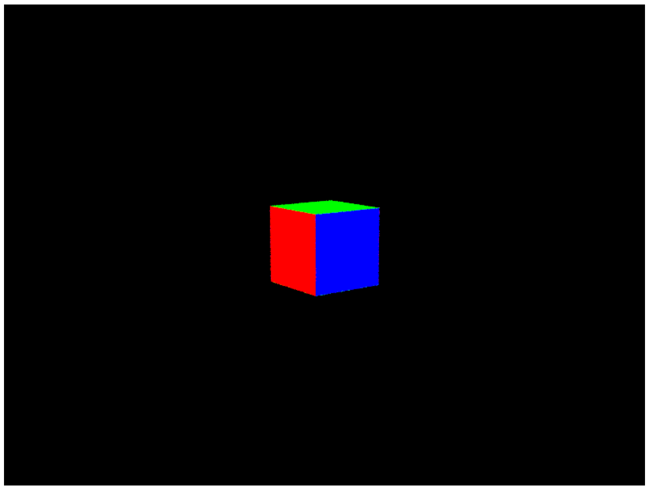


Figure 3: Surface normal approximation of box

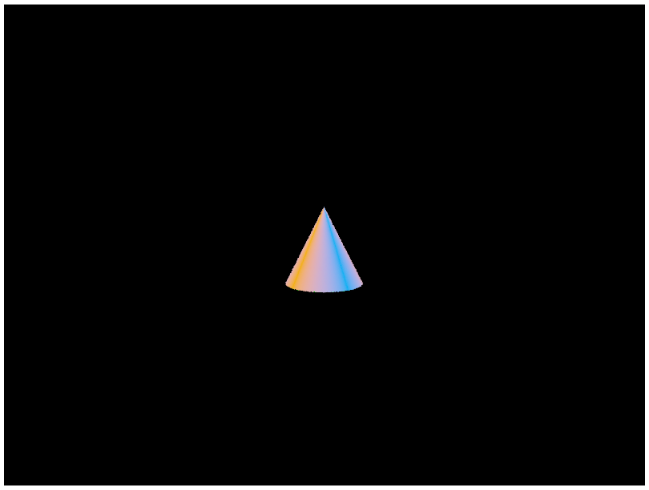


Figure 4: Surface normal approximation of capped cone

2.4 Surface normal approximation

Since signed distance field does not explicitly state the normal vectors along the surface at which $d(p) = 0$, we can approximate the surface normal by taking 6 samples of the signed distance function, two each along the x, y and z directions and find the gradient.

2.5 Constructive solid geometry

Constructive solid geometry is to use Boolean operations for object construction. The basic Boolean operations include union, intersection and difference. We use the following operations to manipulate with different primitives to model a castle.

2.5.1 Union. To find the union of two objects, we simply take the minimum of two distances d_1, d_2, where $d_1 = d_1(p)$, $d_2 = d_2(p)$.

2.5.2 Intersection. The intersection of two objects is the maximum among d_1 and d_2.

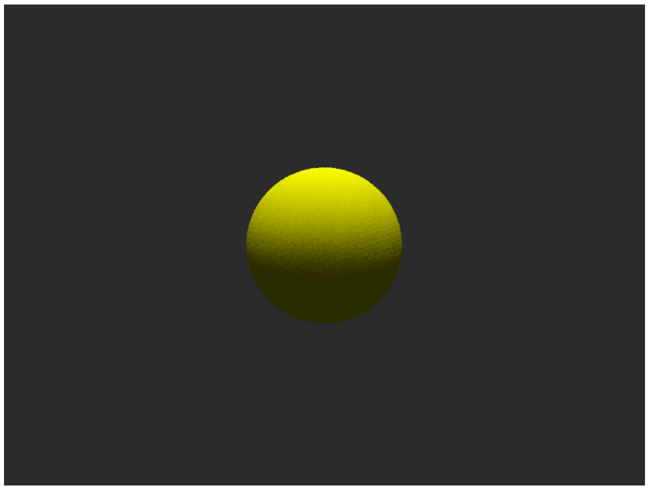


Figure 5: Diffuse BSDF sphere

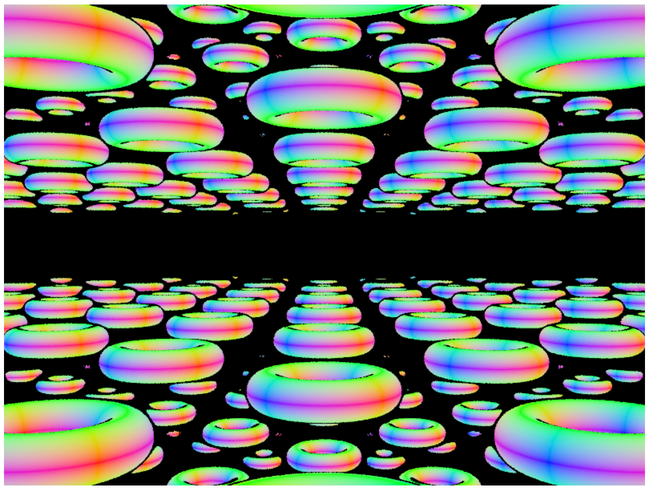


Figure 6: Modeling spatially repeating toruses

2.5.3 Subtraction. Take advantage of the fact that the complement of a signed distance function $d(p)$ is $-d(p)$, and we could decompose subtraction as the intersection of $d_1(p)$ and $-d_2(p)$. So the signed distance function for subtraction is $max(d_1, -d_2)$.

2.6 Domain operations and distance operations

We could take advantage of the mathematical simplicity of implicit modeling to perform a number of domain and distance operations.

2.6.1 Rotation. Rotation can be achieved by applying a rotation matrix which rotates the coordinate space. Rotation preserves the Euclidean norm.

$$d_{\text{rotated}}(p) = d(R * p) \tag{6}$$

2.6.2 Translation. Translation can be achieved by adding a vector to p. Translation also preserves Euclidean norm.

$$d_{\text{translated}} = d(v + p) \tag{7}$$

2.6.3 Repetition. Infinite repetition can be done easily using the modulo operation. Let $a = (a_x, a_y, a_z)$ be the cell size, then repetition with respect to a is

$$d_{\text{repetition}} = d(\begin{bmatrix} p_x + \frac{a_x}{2} & \bmod a_x + \frac{a_x}{2} \\ p_y + \frac{a_y}{2} & \bmod a_y + \frac{a_y}{2} \\ p_x + \frac{a_z}{2} & \bmod a_z + \frac{a_z}{2} \end{bmatrix}) \tag{8}$$

2.7 Menger sponge

Menger sponge is a fractal curve. Mathematically it is defined as the following:

$$M := \bigcap_{n\in\mathbb{N}} M_n \tag{9}$$

, where M_0 is the unit cube and $M_{n+1} := \{(x, y, z) \in \mathbb{R} : \exists i, j, k \in \{0, 1, 2\} : (3x-i, 3y-j, 3z-k) \in M_n$ and at most one of i, j, k is equal to $1\}$
The algorithm to obtain a menger sponge is the following:

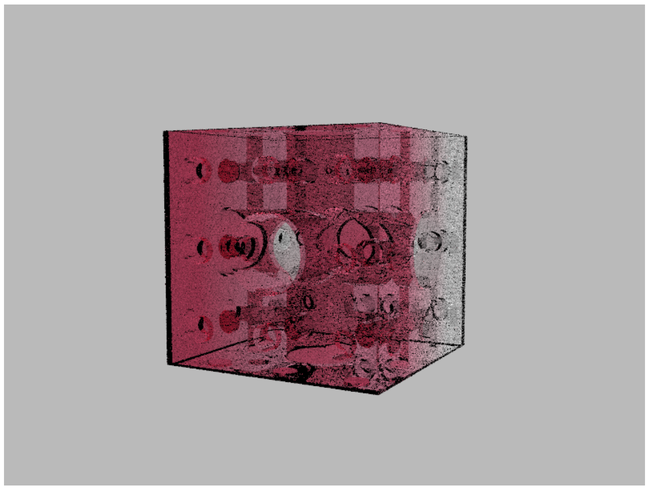


Figure 7: A glass menger box

(1) Begin with a cube
(2) Divide every face of the cube into 9 squares. This will subdivide the cube into 27 smaller cubes.
(3) Remove the smaller cube in the middle of each face, and remove the smaller cube in the very center of the larger cube, leaving 20 smaller cubes.This is a level-1 menger sponge which resembles a void cube.
(4) Repeat steps 2 and 3 for each of the remaining smaller cubes, and continue to iterate ad infinitum. [2]

We combine menger box with other primitives in the castle model to increase the complexity of the scene.

2.8 Bilateral filtering

A bilateral filter is a non-linear, edge-preserving, and noise-reducing smoothing filter for images. It replaces the intensity of each pixel with a weighted average of intensity values from nearby pixels. This weight can be based on a Gaussian distribution [1].

The bilateral filter is defined as:

$$I^{\text{filtered}}(x) = \frac{1}{W_p} \sum_{x_i \in \Omega} I(x_i) f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|) \quad (10)$$

, where the normalization term

$$W_p = \sum_{x_i \in \Omega} f_r(\|I(x_i) - I(x)\|) g_s(\|x_i - x\|)$$

ensures that the filter preserves image energy and I^{filtered} is the filtered image;
I is the original input image to be filtered;
x are the coordinates of the current pixel to be filtered;
Ω is the window centered in x;
f_r is the range kernel for smoothing differences in intensities (this function can be a Gaussian function);
g_s is the spatial kernel for smoothing differences in coordinates (this function can be a Gaussian function)[1].

2.9 Randomizing numbers in GLSL

Last but not the least, one important detail that lies in random sampling procedure is random numbers. To generate random numbers between 0 and 1 we utilize a random procedural texture. The first number generated is from the texture coordinate corresponding to the screen pixel coordinate; the second number and those onwards are retrieved from a slight accumulated offset from that origin. Even though this procedure does not guarantee a uniform distribution, it is workable enough to produce visually compelling results.

3 RESULTS

3.1 Surface normal approximation

We use colors to represent the surface normal: vectors along $(1, 0, 0)$ are colored red, $(0, 1, 0)$ colored green, and $(0, 0, 1)$ colored blue. Each normal vector will be a linear combination of those three colors. The normal vector direction for each point in the scene could be easily visualized using a combination of those colors. Figure 1, 2, and 3 represent the surface normal drawn for a sphere, a box and a cone.

3.2 Global illumination with BSDF

Figure 5 shows a sphere with diffuse bsdf.

3.3 Modeling Spatially Repeating Pattern

Figure 5 shows an infinite repetition of torus. The colors represent normal directions.

3.4 Menger sponge

Figure 7 shows a glass menger sponge with two layers of recursion. Using the x-coordinate of each point, we can bilinearly interpolate the color between red and grey.

3.5 Scene with mixed materials

Figure 8 shows a scene with mixed material. The blue ball is of mirror material, the yellow torus and green cone are of glass material, and the red box is of diffuse material. Note the refraction in the red

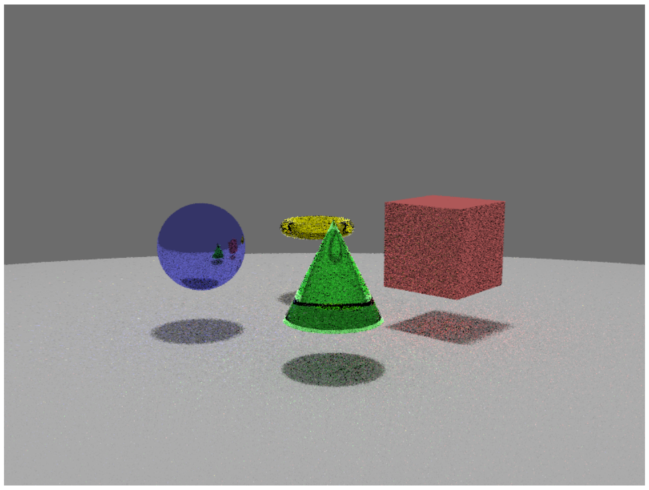


Figure 8: A scene with diffuse, mirror and glass material

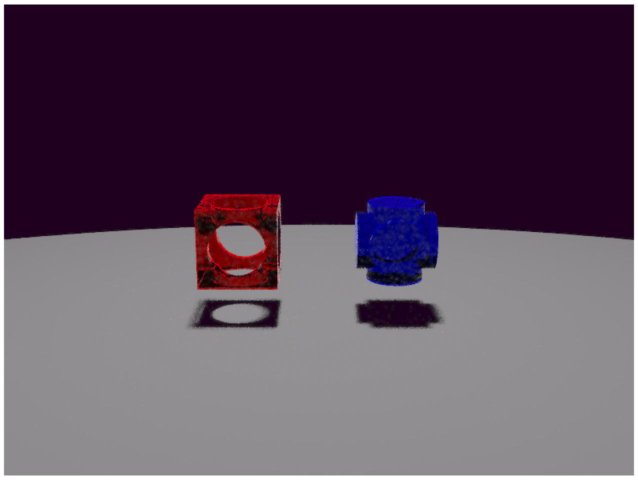


Figure 9: Time varying scene

cone and the reflection of the green cone and red box on the blue sphere.

3.6 Time varying scene

Figure 9 and 10 shows a time varying scene. A time varying scene can be constructed as a linear interpolation between two objects. Let sdf_1 represents the signed distance function of object 1, sdf_2 represents the signed distance function of object 2, a time varying scene can be constructed as $t * sdf_1 + (1 - t) * sdf_2$

3.7 Modeling a complex scene — castle

In order to model the castle tower, we take the difference between a torus and a cone, then elongate along the y axis to get the slender tower top. The top is then taken union with a cylinder. Similarly, for the tower with sharp top, we use four planes to intersect a box, then union it with another box. The whole scene is modeled using primitives including boxes, cones, cylinders, triangle prisms and

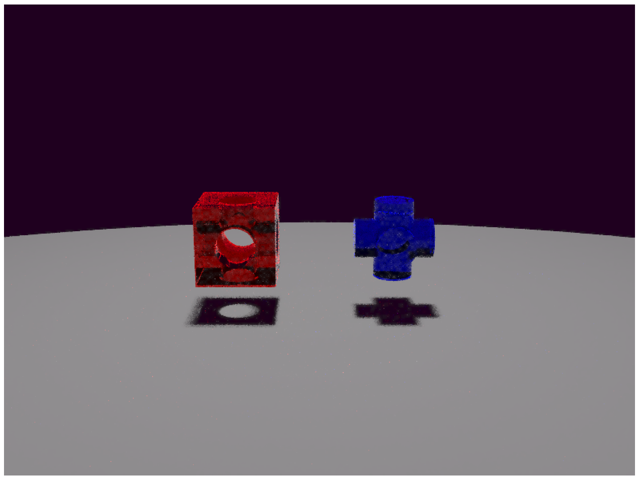


Figure 10: Time varying scene

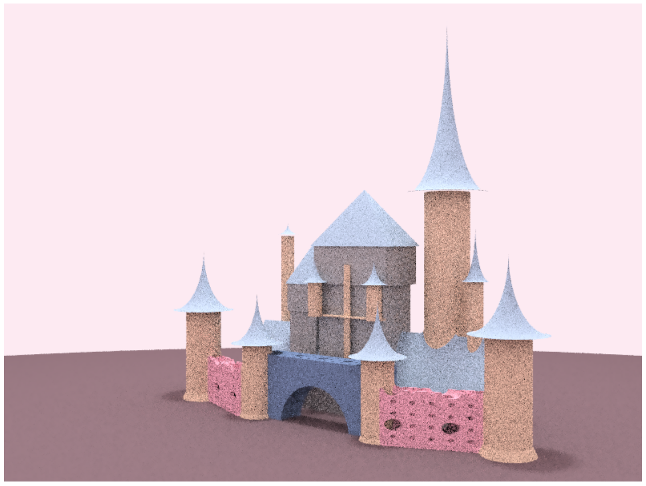


Figure 11: Final rendering of castle

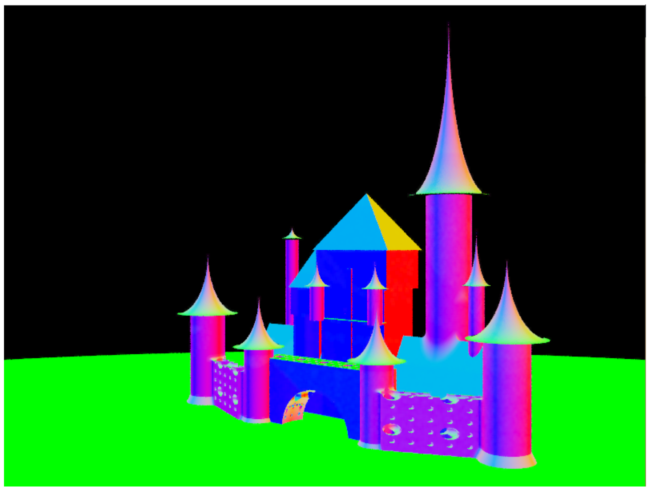


Figure 12: Castle with color representing normal vector

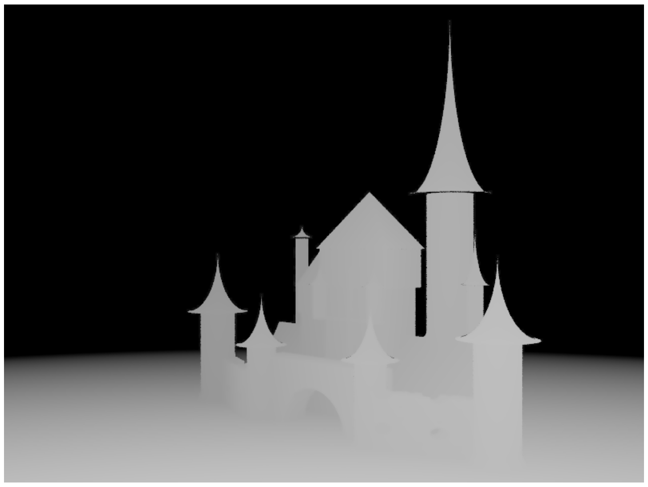
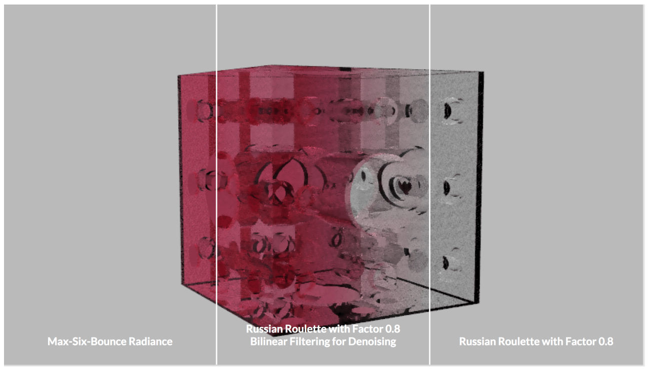


Figure 13: Castle with color representing ray depth


Max-Six-Bounce Radiance
Russian Roulette with Factor 0.8
Bilinear Filtering for Denoising
Russian Roulette with Factor 0.8
Figure 14: Denoised scene comparison

transformations including rotation, translation, reflection. Figure 11 shows the castle with diffuse material. Figure 12 shows the castle with color being the normal of each point. Figure 13 shows the castle with the colors representing depth of ray from the ray marching algorithm. Darker color represent more steps. As can be seen from figure 14, the objects and the ground have significantly lighter color than the background. This is reasonable as ray marching algorithm terminates after max number of steps.

3.8 Denoising with bilateral filter

Figure 11 shows a comparison between using maximum six bounce of radiance and its denoised result with a 5x5 bilateral filter with $\sigma_{\text{Gaussian}} = 10.0$ and $\sigma_{\text{bilateral}} = 0.06$. The noise from Monte Carlo estimation can be slightly reduced with the introduction of such filter. However, there are certain traces of patches of dark spots; it is likely that they are resulted from the random number generator that implicitly creates hidden patterns.

4 CONCLUSION

Implicit surface modeling significantly saves the space to store the scene. In order to render the castle scene in real-time, computer with powerful GPU computation power is needed. Currently we are using an Alienware R17 computer with GTX 1070 graphics card. The best real-time simulation on this computer could run at fps 5 with pixel sampling rate 8, light sampling rate 2, max number of bounces 3 and GPU usage 100 percent. We failed to get a better performance as increasing any of those sampling parameters will cause memory overflow.

REFERENCES

[1] [n. d.]. Bilateral filter. Retrieved May 2, 2018 from https://en.wikipedia.org/wiki/Bilateral_filter
[2] [n. d.]. Menger sponge. Retrieved May 2, 2018 from https://en.wikipedia.org/wiki/Menger_sponge
[3] Carl Lorenz Diener. 2012. Procedural modeling with signed distance functions.
[4] Inigo Quilez. [n. d.]. Modeling with distance functions. Retrieved April 23, 2018 from http://iquilezles.org/www/articles/distfunctions/distfunctions.htm
[5] Jamie Wong. 2016. Ray Marching and Signed Distance Functions. Retrieved April 23, 2018 from http://jamie-wong.com/2016/07/15/ray-marching-signed-distance-functions/