Computational Graphics, Homework 2
Progressive Photon Mapping

Kefan Dong
2016011272
kefandong@gmail.com

Notice: Bugs in Bezier curve are fixed after the last meeting, and anti-aliasing is added.

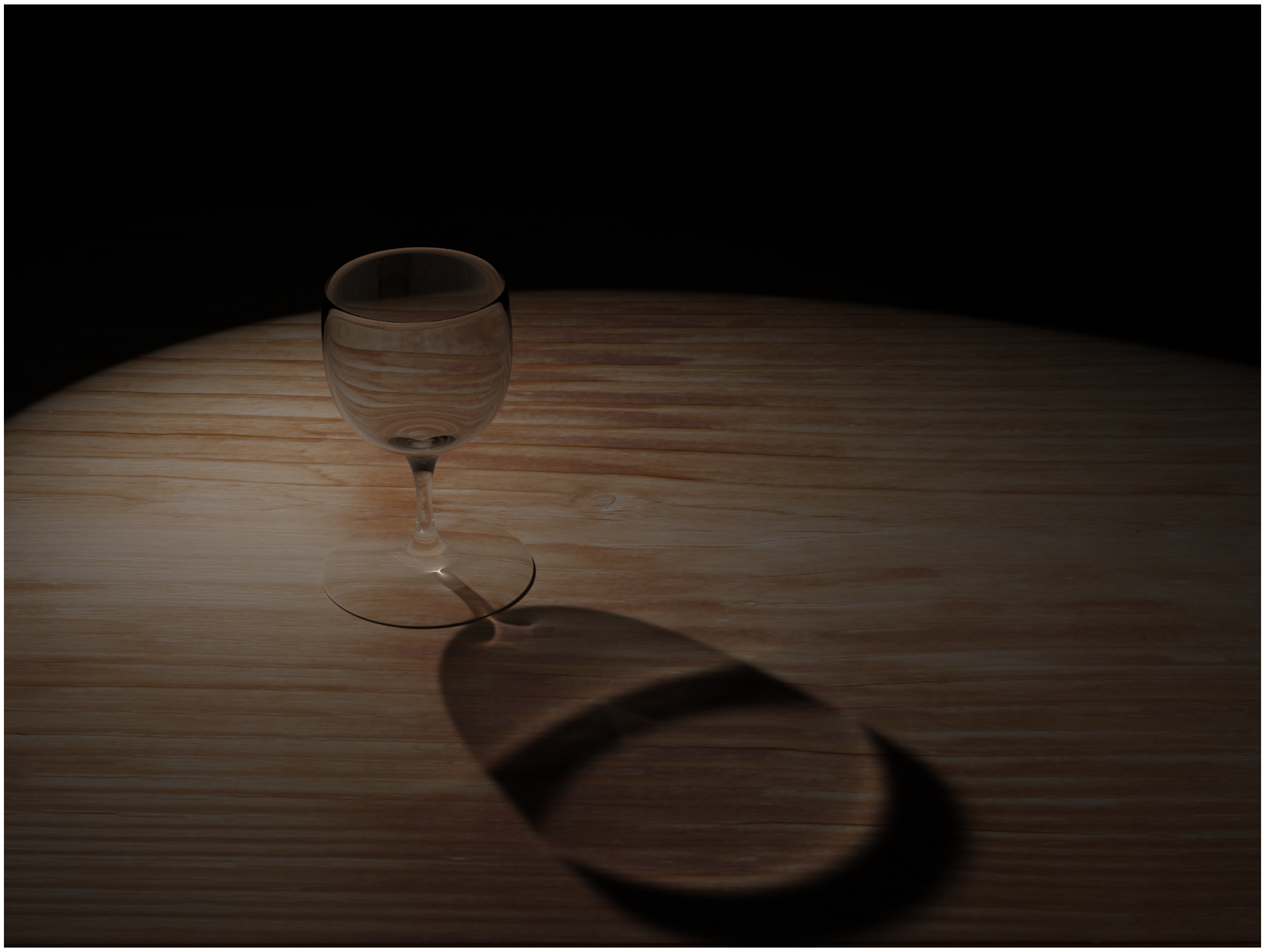

Figure 1: The final output image. For the full image please see *final.png*.

1 Algorithm

The algorithm is base on the progressive photon mapping methpd proposed by Hachisuka *et al.*[1].

1.1 Hitpoint

Tow methods are implemented in this project for hitpoint query. The first method is KD-tree and the second method is Hash table. The final version of the implementation is Hash table since its more efficient. We compare the pros and cons of these two methods below.

- KD-Tree.
  - Parameter free: The construction of KD-Tree does not require any hyper-parameter and therefore more robust.
  - Constant memory: The construction of KD-Tree requires no additional memory.
- Hash table.
  - Fast and simple: by choosing proper hyper-parameter (i.e., the radius for discretization), Hash table can be at least twice faster than KD-Tree. In this implementation the space is cut into cubes with side length twice the initial radius of PPM method.
  - Additional memory: Since the hit points are discretized in a periodic manner (i.e., the space is cut into regular cubes), either 4 queries are needed for a photon, or every hit point is added to 4 cubes. This implementation adapts the latter method since for high quality images, the number of photons is significantly larger than hit points.

Note that when querying points inside a ball, both KD-Tree and Hash table have no theoretical guarantee. The $\mathcal{O}\left(n^{2/3}\right)$ running time analysis for KD-Tree only applies when querying points inside a cube. Furthermore, both method can not exploit the fact that the radius of hit points are decreasing.

Implementation of KD-Tree and Hash table is in *utils.hpp*, form line 206 to line 352.

1.2 Bezier curve

We implement a Bezier curve rotation body, i.e., the body is generated by a Bezier curve rotating along an axis. For example, the following curve is the Bezier curve for a wine cup.

The rotation body of Bezier curve can be represented by the parametric equation

$$\begin{cases} x = x_0 + u(t)\cos\theta, \\ y = y_0 + v(t), \\ z = z_0 + u(t)\sin\theta. \end{cases} \tag{1}$$

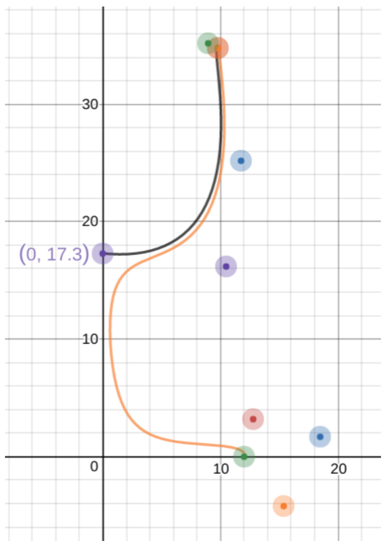

Figure 2: Bezier curve for wine cup.

for $t \in [0,1], \theta \in [0, 2\pi]$. For an n-order Bezier curve, $u(t)$ and $v(t)$ are polynomial of order n. And a ray can be represented as

$$\begin{cases} x = s_x + kv_x, \\ y = s_y + kv_y, \\ z = s_z + kv_z. \end{cases} \tag{2}$$

Eliminating y, it follows that

$$\begin{cases} x = s_x + \frac{y-s_y}{v_y} v_x, \\ z = s_z + \frac{y-s_y}{v_y} v_z. \end{cases} \tag{3}$$

From equation (1) and (3), we have

$$\begin{cases} s_x + v_x \frac{y_0+v(t)-s_y}{v_y} = x_0 + u(t)\cos\theta, \\ s_z + v_z \frac{y_0+v(t)-s_y}{v_y} = z_0 + u(t)\sin\theta, \end{cases} \tag{4}$$

and therefore

$$(s_x + v_x \frac{y_0 + v(t) - s_y}{v_y} - x_0)^2 + (s_z + v_z \frac{y_0 + v(t) - s_y}{v_y} - z_0)^2 = u(t)^2. \tag{5}$$

Since $u(t), v(t)$ are polynomials of order n, equation (5) is a polynomial equation with order at most $2n$. In this implementation we solve the intersection of Bezier rotation body and a ray by solving polynomial equation (5).

For a polynomial equation $f(t) = \sum_{i=0}^{2n} a_i t^i$, we can find one of its root t^* by Newton's iteration. and then we recursively solve $f(t)/(t - t^*)$. The polynomial division can be calculated efficiently. Therefore by this recursion, all roots for $f(t) = 0$ could be solved, except for those who can not be found by Newton's iteration. And then we choose the solution that is nearest to the starting point of the ray as intersection.

Although solving the root of a polynomial is quiet robust, for some cases Newton's iteration can not find certain solution if the start point of iteration is not good enough. Here we use an heuristic method to choose the start point of Newton's iteration. The idea is to approximate the Bezier curve by piecewise linear function. Note that for a linear function, equation (5) has order 2, which can be solved analytically. The piecewise linear function is chosen as follows. For a Bezier curve $u(t), v(t)$, let $t_i = i/M$, for $0 \leq i \leq M$. The piecewise linear function is the function connecting $(u_{t_i}, v_{t_i}), (u_{t_{i+1}}, v_{t_{i+1}})$. Besides, we also choose another piecewise linear function that is constructed by the connecting the intersections of tangent line at point $(u_{t_i}, v_{t_i}), (u_{t_{i+1}}, v_{t_{i+1}}), (u_{t_{i+2}}, v_{t_{i+2}})$. The following figure is an illustration of the approximation.

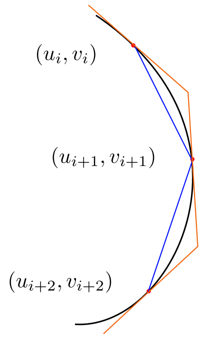

Figure 3: The approximation method. The black curve is the Bezier curve. The blue segments is generated by connecting $(u_{t_i}, v_{t_i}), (u_{t_{i+1}}, v_{t_{i+1}})$, and the orange segments is generated by connecting the intersections of tangent line at point $(u_{t_i}, v_{t_i}), (u_{t_{i+1}}, v_{t_{i+1}}), (u_{t_{i+2}}, v_{t_{i+2}})$.

For a convex Bezier curve, it lies inside these two piecewise linear function. Therefore a ray intersect the Bezier rotation body only if it intersect at least one of the rotation body of these two piecewise linear function. These method actually behaves well enough in the implementation.

Remark. With only one of the approximation method, certain roots can not be found, which leads to black pixels in the image. Notice that this problem can only be revealed when rendering an Bezier rotation body with *glass* material. This is because for diffusion or metal surface, a missing root only causes the ray passing through (i.e., the edge of the body becomes blur). But for glass surface, it causes the ray actually going into the body instead of reflect away (by Fresnel equation). And thereafter the ray can not escape the body without encountering another missing root problem.

The implementation of Bezier function is in *utils.hpp*, from line 353 to line 412. Code for solving equation (5) can be found between line 449 and line 722 of *objects.hpp*.

2 Speed up

2.1 Preprocessing

The performance can be improved significantly by preprocessing and precomputing. For example, to calculate equation (5), one need to calculate the square of several polynomials. However, we can precompute $u(t)^2, u(t)^2$ and then the calculation of equation (5) is nothing but polynomial addition, which is one order faster than polynomial multiplication.

2.2 Openmp

The photon mapping method can be easily implemented in a multi-thread computation way. Here we use library openmp to do the multi-thread programming. We emphasize that implementing an thread-safe program is not trivial, even when using libraries like openmp. For example, the random number generator of C++ (i.e., random device in C++11 standard) is not thread safe, which will cause unexpected artifacts.

3 Results

The final output image is *final.png*. The size of the image is 2400×1800. The image is produced by 8 anti-aliasing round, with $5 * 10^8$ photons each round.

Note that this image is not produced by super-sampling. We can see that the proposed heuristic start point selection method is indeed robust.

References

[1] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. *ACM Trans. Graph.*, 27(5):130:1–130:8, December 2008.