Ray Tracer

Zhu Yue (Ann)
y293zhu
ID: 20646992
Nov 30, 2018

This is the report for my CS488 final project, Ray Tracer. The report including the following sections: purpose(the purpose of the project), statement (a brief introduction of scene and the related Computer Graphics features), technical outline (algorithms to achieve the objects), manual(guide to running the project) and implementations(software design considerations), bibliography and an objective list.

1 Purpose

The purpose of this project is to implement a ray tracer including the features like reflection, soft shadow, etc and create an interesting scene using the implemented ray tracer. Also, try to make use of the hardware to accelerate the program.

2 Statement

My scene is about room inspired by a Japanese animation. In the room, there is a mirror which requires the mirror reflection to be implemented. Also, it is supposed that part of the model which cannot be seen directly, can be found in the mirror. In addition, I put a cup of water on the cabinet, which will use the feature of refraction. Besides, the wall, floor will need texture mapping and bump mapping to look more realistic. Phong shading will be add to each mesh to have a smooth surface. Also, each object is supposed to cast a soft shadow for realistic purpose, and for the transparent object, e.g. the cup of water, the shadow casted by the cup of water should by "lighter" than the shadow casted by nontransparent object. Extra primitives may be used for small object like chopsticks.

3 Technical outline

- **Extra primitives**:
  Use the formula to detect intersection:
  Cylinder: $(x - x_0)^2 + (z - z_0)^2 = radius^2$
  Cone: $(x - x_0)^2 + (z - z_0)^2 = (\frac{z - z_0}{height})^2$
  Torus: $((x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 + R^2 - r^2)^2 = 4R^2(x^2 + y^2)$

- **Bump mapping**:
  Decide the uv coordinate for each object, and map the coordinate to the normal mapping image to get a normal. For normal vectors from

mapping image is pointing the positive z-axis, however, the origin normal vector may not point to the z-axis. Therefore, we need to calculate the angle between the origin normal and the z-axis, and then rotate new normal vector by that angle.

- **Multithreading**:
  Use the thread library in the C++, and divide the image into small blocks. Each thread only render one block each time. As soon as the block render has completed, the thread should fetch a new block to render until all the blocks have been rendered.

- **Super sampling**:
  Consider each pixel as a little square, and then uniformly get points on that square. Calculate the ray go through each point and then calculate the mean to get the final color.

- **Phong shading**:
  Required to calculate the vertex normal before rendering, if the vertex normal is not been given. The vertex normal can be calculated easily by accumulated all the polygon normals where the vertex is in. After calculating the vertex normal, the point normal is the result of Barycentric Interpolation using vertex normals.

- **Reflection**:
  The reflect out ray can be calculate by the formula:
  $reflectout = 2(normal \cdot reflectin)normal - reflectin$
  where both normal and reflectin is the unit vector. And the final colour should be the $factor * colour1 + (1 - factor) * colour2$, colour1 is shaded at the reflective object and colour2 is shaded where the reflectout ray hits.

- **Refraction**:
  Let θ_1 be the angle between refractin ray and normal and θ_2 be the angle between refractout ray and indices of two medias are n_1, n_2, then $\frac{sin\theta_1}{sin\theta_2} = \frac{n_2}{n_1}$
  Therefore, the angle between refracin ray and refractout ray should be the $\frac{\pi}{2} - \theta_1 + \theta_2$, and the rotation axis can be calculated by the cross product of refractin ray and normal vector.
  Recursively calculate the refractout ray until the ray hits a nontransparent object.

- **Soft shadow**:
  One way and the way I used in my project to implement soft shadow is

to use area light instead of point light. Uniformly pick some points in the area light and detect whether it can reach the point directly. And finally, divide by how many rays we cast from the area light.

- **Depth of field**:
  Use formula $C = |A\frac{F(P-D)}{D(P-F)}|$ to calculate the diameter for circle of confusion on the scene, where A is the aperture radius, P is the plane in focus, D is the object distance and F is focal length.Then calculate the mean of colours shaded by each ray go from the point in the circle and through focal point.

4 Manual

- **Lua script**(including all the accepted lua command in the program)
  The following commands are the same as in A3 and A4
  1. **gr.node(name)**:Return a node with the given name, with the identity as its transformation matrix and no children.
  2. **gr.mesh(id,name)**: Return a GeometryNode tied to the mesh with the given ID, and with the given name. **All the models should be put at ./Project/Model/ directory**
  3. **gr.material({dr, dg, db}, {sr, sg, sb}, p)**: Return a material with diffuse reflection coefficients dr, dg, db, specular reflection coefficients sr, sg, sb, and Phong coefficient p.
  4. **node:rotate(axis, angle)**: Rotate node about a local coordinate axis ('x', 'y' or 'z') by angle (in degrees).
  5. **node:translate(dx, dy, dz)**: Translate the given node by (dx, dy, dz).
  6. **node:scale(sx, sy, sz)**: Scale node by (sx, sy, sz).
  7. **gr.nh_box(name, {x,y,z}, r)**: Create a box of the given name.
  8. **gr.nh_sphere(name, {x,y,z}, r)**: Create a sphere of the given name, with centre (x, y, z) and radius r.

  The following commands are the different from A3 and A4 or new in Project
  9. **gr.nh_cylinder(name, {x,y,z}, h, r)**: Create a cylinder of the given name, with center (x,y,z), height h and radius r.
  10. **gr.nh_cone(name, {x,y,z}, h, r)**: Create a cone of the given name, with center (x,y,z), height h and radius r.

11. **gr.nh_torus(name, {x,y,z}, R, r)**: Create a torus of the given name, with center (x,y,z), radius R and r.
12. **gr.texture(filepath, {sr, sg, sb}, p)**: Return a texture with specular reflection coefficients sr, sg, sb, and Phong coefficient p. **All the textures should be put at ./Project/Texture/ directory**
13. **gr.bumpmap(filepath)**: Return a bumpap at the filepath. **All the bumpmaps should be put at ./Project/Bump/ directory**
14. **node:set_refraction(i, t)**: Set the media index and the transparency factor of the geometry node.
15. **node:set_reflection(t)**: Set the reflection factor for the geometry node.
16. **node:add_child(c, (m), (t), (b))**: Add the child node to a scene node. If c is a geometry node, then require to set material, texture, and bumpmap for c as well. Can set t and b to nil if no need to add texture and bumpmap for the node.
17. **gr.focal(p,f,a)**: Create a focal len with the focus at p, the focal length a and the aperture radius a.
18. **gr.light(x,y,z, r,g,b, c0,c1,c2, r, brightness)**: Create a area light source with radius r located at (x, y, z) with intensity (r, g, b) and quadratic attenuation parameters (c0, c1 and c2), also brightness to set fmax(dot(normal, l), brightness).
19. **gr.render(node, filename, w, h, eye, view, up, fov, ambient, lights, focallen)**: Raytrace an image of size wh and store the image in a PNG file with the given filename. The parameters eye, view, up and fov control the camera, with fov being the field of view in the y-direction (up direction). The last three parameters are the global ambient lighting , a list of are light sources in the scene and a focal len.

- **Compile**: At the directory ./Project
  $ premake4 gmake
  $ make

- **System dependence**: The program compile for different OS system(Mac OS, Linux and Windows) is already handled by the premake4 gmake command. However, the project requires the opengl, lodepng and lua to be pre-installed.

- **Running command**: to render the scene stored in the lua script, the default filepath is project.lua
  $ ./Project (project.lua)

- **Output**: At first it will display the biggest number of threads the machine can support, and output the process status by one percent. After the render has been done, it will show "Render Complete!" info at the end. The rendered scene will be stored at directory ./Project

- **File in the directory**

  1. **./Blender**: All the blender files used to create mesh file.
  2. **./Bump**: Store all the bump mapping image files.
  3. **./Model**: Store all the mesh (obj files)
  4. **./Texture**: All the texture images for the geometry node.
  5. **./Documentation**: Store all the sample images for each objective and the report, demo file for the project.
  6. **Source code**: All the source code are put in the ./Project directory

5 Implementations

In general, I use the VS code to write my project on Mac OS and use the installed lldb to debug for the project. Also, in some cases, I output the information I need to debug into a testfile for figuring out the problem.

- **Data Structure and Algorithm**
  Since the final project code was extended from the Assignment, to go through all the nodes, I just use a multiple children tree as in A3 and A4.
  Added a material map to find all the created mateiral/texture/bumpmap. And after reading the lua script, all the materials pointers are supposed to stored in the material map and each geometry node can get a key to fetch the material. While the rendering process asks for material information, it can directly look up in the material map use the stored key. And after the render process, materialmap can be used to clear all the date for materials.
  A 2D vec3 vector: imageBuffer is stored in the Render class, and is used to memory the information of the super sampling pixels.

For the mesh, use 7 1D vectors to store all the information: m_vetives(vec3 vector): position information, m_uvInfo(vec2 vector): uv coordinates, m_normals(vec3 vector): normal for triangle faces, m_VetexNormals(vec3 vector): normal for vertices, m_vNormals(Triangle vector):index to find vertex normal for each triangle face, m_vUVs(Triangle vector): index to find uv coordinates for vertex and m_faces(Triangle vector): index to find position for triangle face's vertex
Algorithm used to implement objectives are exactly the same as described in the **technical outline**.

- **Code Map**

  1. **Extra primitives**: To calculate the coefficient for the torus formula, I write a simple polynomial add, multiple, scale function in the 'PolyCal.cpp' and all the intersection function is in 'Primitive.cpp' and 'Mesh.cpp' called *intersect*
  2. **Bump mapping**: All the uv coordinates are defined in the *intersect* function, and the new normal rotation is also completed in that function.
  3. **Multithreading**:Multithreading is implemented in the file 'Project.cpp'. *renderImage* is used to create multiple thread and delete the data after render process. *asyncThread* is used for each thread to fetch block and render the block. *renderImageThread* is used for render each block.
  4. **Super sampling**:*renderImage* is responsible for computing the super sampling information into the required size. And *renderImageThread* is responsible for cast multiple ray from one pixel. How many rays to calculate one pixel is decided by the global constant pixelCount in 'Project.cpp'
  5. **Phong shading**:Barycentric Interpolation is used in the *intersect* funtion in 'Mesh.cpp' and I also implement extra helper function *calculateArea* to calculate the area for a triangle in the same file.
  6. **Reflection**:reflectout ray is calculated by the function *reflectOut* in the 'Ray.cpp' and the final return color for the reflect ray is handled by *calculateReflection* in the same file.
  7. **Refraction**:refractout ray is calculated by the function *refractOut* in the 'Ray.cpp' and the recursive call to find the hit object and final return color for the refract ray is handled by *calculateRefraction* in the same file.

8. **Soft shadow**: *calculateShadowRay* in the 'Ray.cpp' is responsible for cast mutiple shadow ray from the area light and calculate the soft shadow
9. **Depth of field**: The radius of circle of confusion is calculated in *renderImageThread* in 'Project.cpp' after finding the interection point.

- **Potential Bugs**

1. For refraction, didn't handle the situation that when the angle is bigger than the critical angle, then a total internal reflection appears instead of refraction.
2. For depth of field, I set the limit of the radius of circle of confusion, however, the limit should be set by some formula for the "infinite far" object.
3. Didn't detect whether the material, texture, bumpmap is a nullptr when they are added to the geometry node. Therefore, when try to access these non-exist materials, a segmentation fault will appear.

- **Future Possibilities**

1. Fix the bug in the refraction and depth of field part.
2. Use the adaptive anti-aliasing(edge detect) or spatial division to speed up.(Currently time for render the project scene in 1280*720 with soft shadow, super sampling(9 rays/pixel) and depth of field feature will need around 10 hours on a 64 thread computer)
3. Add the notification when the user try to add a not exist material to a geometry node.
4. I'd like to try implement photon mapping, especially for the caustic for water or other transparent object in the future
5. I tried to implement the soft shadow by using the shadow map and then apply the Gaussian distribution multiple times on the shadow map. Although it can give a good effect for a simple scene, it can't handle the reflective and refractive very well, since it will requires to store information not only for object that primary ray hits but also the objects that secondary ray hits. I'd like to find a way to handle the reflective and refractive object.

- **Other Acknowledgement**

The model for the cup, texture images and bumpmap images were

downloaded from the Internet, while other models are created by my self.

6 Bibliography

Woo, A., Poulin, P. and Fournier, A. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, Vol.10(6), pp.13-32, Nov. 1990.

Demers, J. (2004). *GPU Gems* - Chapter 23. Depth of Field: A Survey of Techniques. Retrieved February/March, 2016, from `http://developer.download.nvidia.com/books/HTML/gpugems/gpugems_ch23.html`

Brabec, S. ; Seidel, H.-P. Hardware-accelerated rendering of antialiased shadows with shadow maps.*Computer Graphics International 2001*, Proceedings [1530-1052],pp:209 -214, 2001

7 Objectives

Full UserID:y293zhu Student ID:20646992

___ 1: Objective one. Extra primitives has been implemented. (cylinder, torus, cone)

___ 2: Objective two. Bump mapping has been added.

___ 3: Objective three. Use mutithreading for acceleration.

___ 4: Objective four. Super sampling is used for anti-aliasing.

___ 5: Objective five. Phong shading has been added for the model.

___ 6: Objective six. Reflection has been implemented for the mirror.

___ 7: Objective seven. Refraction is added for the cup of water.

___ 8: Objective eight. Soft shadow has been added.

___ 9: Objective nine. Depth of field has been implemented.

___ 10: Objective ten. My final rendered scene.

A4 extra objective: Texture mapping for sphere/cube/mesh triangle