Ray Tracing Renderer: Reflection Technique for Game

Anting Zhou

Abstract—**Game is become an important part of our life. Many graphics professors focus on providing a realistic virtual world. One of the focuses is reflection technology. This article will introduce different techniques for rendering the reflection and analyze their advantages and disadvantages. In those techniques, this article will introduce the ray tracing renderer in detail and generally design the classes which will be used in ray tracing renderer. The process design also is introduce in brief and describe how to perform basic ray tracing technique.**

***Index Terms*—Environment mapping, Ray Tracing, Reflection, Rendering Technique, Screen-Space Reflection**

I. Introduction

RENDERING technology is developed with game industry for many years. Developers are try their best to provide highly realistic or even surreal virtual environment. But still, there lots of technical challenges in this area, like the simulation of water or smoke, ambient occlusion and high quality reflection. Reflection phenomenon can be seen everywhere in real world but hard for computer simulation in real time. The contradiction of efficiency and accuracy confuses the developer all the time in this area. Screen-space reflection and environment mapping are two kinds of implementation generally used in modern games. Their both simplified the rendering progress by some techniques with the cost of providing a inaccurate result. And an other solution is ray tracing which simulate the refraction and reflection of the ray of light like the real world. It provides high accuracy of reflection image but increase the performance costs of graphic card.

II. Environment mapping

Environment mapping is a common method used in games because of its low difficult of implementation. In brief summary, environment mapping generates a cube map of environment based on a specific camera. Then the algorithm calculates the reflex direction rely on the incident direction and samples the environment map using reflex direction.[1] The Environment mapping performs acceptable reflection of static and far object with insignificant performance cost because the reflection map rendered only once and can be preprocessed. But for dynamic object, it is hard to perform environment mapping due to the high cost of generating environment map every frame. Also the environment mapping only provides the approximate result and the error become significant when the object is near the reflection surface. The error mainly comes from the difference of reflection position and camera position. As the Fig. 1. shown, the reflex light hits the spot A, but the algorithm samples the environment map by the reflex direction which is equivalent to the color information at spot B.

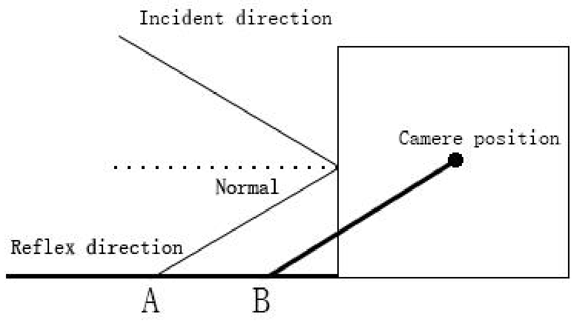


Fig. 1. Error in Environment Mapping

III. Screen-Space reflection

With the increasing requirement of graphics in games, some new techniques come up to provide better performance. SSR(Screen-Space Reflection) is one of the common used techniques. SSR is image-based algorithm which uses the information stored in G-buffer to calculate the reflection data. For each frame, the normal and position information based on screen space will be rendered into G-buffer. When performing reflection, all position like the position of light source will transform to same coordinate then calculate the reflex direction and sample the color buffer[2].

But because the SSR is using the information of G-buffer which is incomplete compared with the whole scene. In some situation, like the calculated reflex ray of light hits the back face of an object which is unseen in the screen space(shown as Fig. 2.), the algorithm will sample nothing and use an alternative solution such as perform blur in that area.

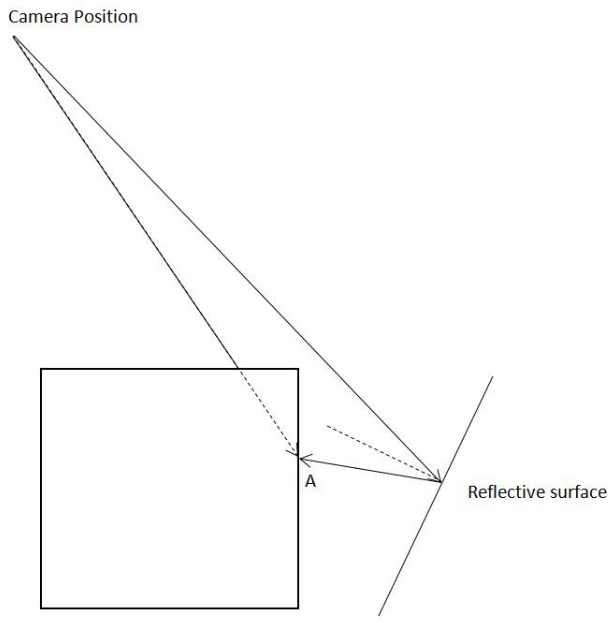


Fig. 2. Disadvantage of SSR

IV. Ray tracing

Ray tracing is a technology that calculates the colour of each pixel by tracing the path of their ray of light and simulating the interaction of virtual object and get the final image as output.[3] Because of following the physical rules of light transmission, ray tracing can get more accurate and more realistic image compared with environment mapping and screen-space reflection mentioned above. And it can also simulate almost all the optical phenomenons like refraction, reflection, scattering and dispersion. But it requires massive calculation which hard to be satisfied by personal computer even their performance improved a lot these years. Thus, ray tracing mainly was used in off-line rendering such as 3D animation or special effects in films. But due to requirement of real-time calculation in computer game area, the application of ray tracing is infrequent.

Ray tracing algorithm is highly parallelized.[4] Tracing the ray of light for each pixel in image can execute in parallel because the calculation of each pixel not rely on the result of others. The calculation amount of ray tracing is direct proportional to the quantity of pixels in output image. Thus, the time-consumption can be reduced remarkably if the ray tracing algorithm can run totally in parallel. When apply parallel computing in particular, for example in rendering 3D film, hundreds of computers render the image in same time dividing by block generally. But for personal computer, because the CPU executes the instructions in series, it is hard to apply parallel computing physically. And if solving the problem by adding the cores in CPU, the cooperation between cores will become complex and restrict the software structure.

Differ from CPU, GPU is designed for image-relevant calculation which provides brilliant parallel computing performance. Due to its logical control ability is relatively weak, the main job of GPU was rendering the vertex and fragment for a long time. But the situation is different these years, because of the development of graphic card. With the increasing demand of parallel computing in personal computer, graphic card manufacturer keep upgrading the technology. The concept of rendering streamline and GPGPU(General Purpose Graphic Process Unit) came up providing acceptable solution for apply parallel computing in personal computer. Also some computing language or describe language are provided by manufacturer for graphic engineers to reduce the difficulty of graphic development. All those changes make it possible to apply complex parallel computing, such as ray tracing, in personal computer and give it acceptable performance.

V. Demand analysis

A. *Diffuse reflection*

The diffuse reflection happens on object surface when directional light lighting rough surface. The light will scatter off the surface making it possible to be seen in all directions. But the diffuse light not uniformly distributed in all directions. It has some certain rules, like Lambert's cosine law, which reveal the correlation between brightness and included angle between surface normal and incident direction. Diffuse light is commonly seen in real world which needs appropriate implementation in renderer.

B. *Shadow*

When the light from the light source is blocked by opacity object, dark area will be left as shadow. The performance of the shadow is a important part in graphic rendering, which also is a hard part. The shadow in real world is complex because of the multiple light source and indirect illumination. So, how to provide a realistic and accurate shadow is a focus of graphics. This article will mainly focus on reflection and ray tracing technique itself. So this article only talks about the shadow produced directly block.

C. *Specular reflection*

The specular reflection occurs when the ray of light intersects with smooth surface. Specular reflection follows the Law of Reflection that incident angle equals to reflex angle, and they are coplanar to normal.[5] Thus, the specular light only can be observed in certain direction. The common representation of specular reflection is rendering the combined image of object surface and reflected image based on the smoothness of the surface. Specular reflection is generally used in modern game in which ray tracing algorithm provides more realistic performance than environment mapping and screen-space reflection.

D. *Refraction*

The light will be refracted when it intersects with the interface between two transparent materials with different refraction index. The refraction follow the Snell's Law which describes the correlation between the included angles that between incident light, refraction light and normal interface normal. Compared with other kinds of renderer, ray tracing can easily implements complex refraction phenomenons like convex lens and concave lens imaging by simulating the propagation of light.

E. *Recursive ray tracing*

As introduced above, when the light hits the surface, it may generates two rays of light based on the refraction and reflection. A ray of shadow may also be generated depends on the transparency of the material it hits. The rays of refraction and reflection light may also hit the surface and generate more rays. Thus, the process of solving the ray tracing should be recursive to solving those rays which provides high degree of realism.

VI. OUTLINE DESIGN

A. *Class designs*

The ray tracing renderer is designed as Fig. 3. based on the features of ray tracing algorithm.

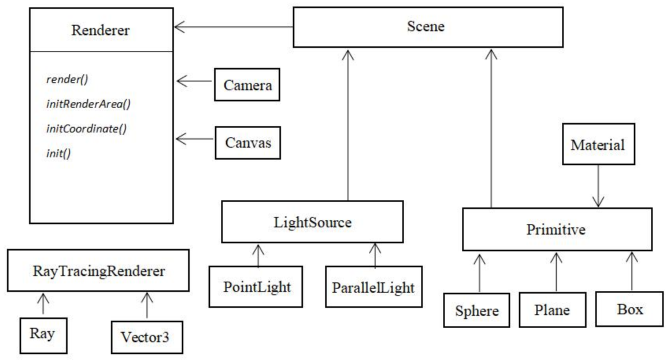


Fig. 3. Classes in Ray Tracing Renderer

1) *Vector3*

Vector3 is a basic data structure in ray tracing renderer. Because this article is going to implement ray tracing a 3D render technique, all calculation of ray tracing is done in 3D space. The position and vector in 3D space are described by x, y, z components. When solving the space geometric problem, normalization, dot product, cross product are generally used. So it is important to implement those method and override the add and minus operator.

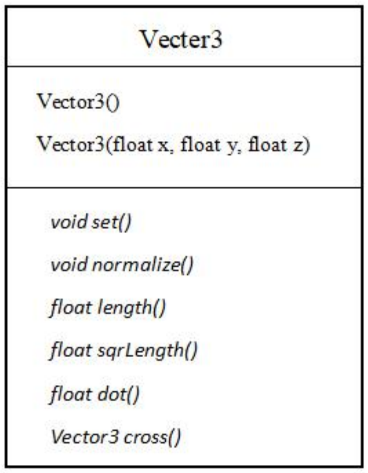


Fig. 4. Vector3 Class

2) *Ray*

Ray class represent a ray of light which include the origin and its direction normal. Those information are used for ray-object intersection to figure out the intersection point.

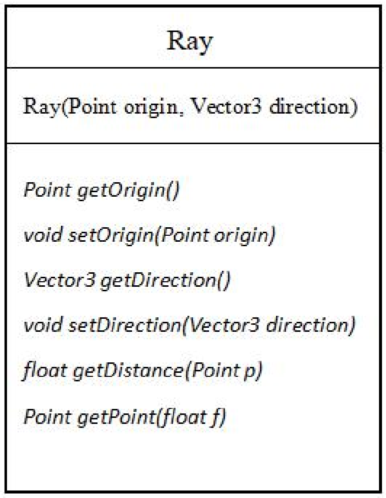

Ray
Ray(Point origin, Vector3 direction)
Point getOrigin()
void setOrigin(Point origin)
Vector3 getDirection()
void setDirection(Vector3 direction)
float getDistance(Point p)
Point getPoint(float f)

Fig. 5. Ray Class

3) *Camera*

Camera class is used to stored status of the camera, which represent the observer, include its position, pitch and yew which describe the observation angle and projection matrix which giving the information of near and far plane, field of vision and etc. For free observation, the camera class also need implement move and rotate method. When preforming ray tracing, the original ray is build by connecting camera position and pixel in viewport.

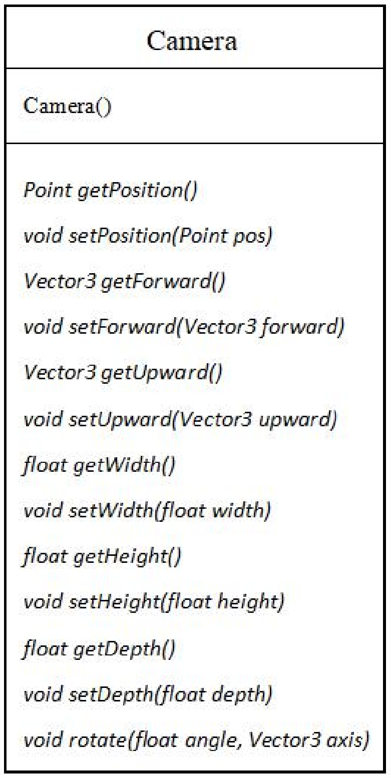

Camera
Camera()
Point getPosition()
void setPosition(Point pos)
Vector3 getForward()
void setForward(Vector3 forward)
Vector3 getUpward()
void setUpward(Vector3 upward)
float getWidth()
void setWidth(float width)
float getHeight()
void setHeight(float height)
float getDepth()
void setDepth(float depth)
void rotate(float angle, Vector3 axis)

Fig. 6. Camera Class

4) Canvas

Canvas class represent the output of the renderer. Color information of pixels may render to frame buffer or screen. Those configuration recorded by canvas, so that when output is changed, only canvas need to be modified.

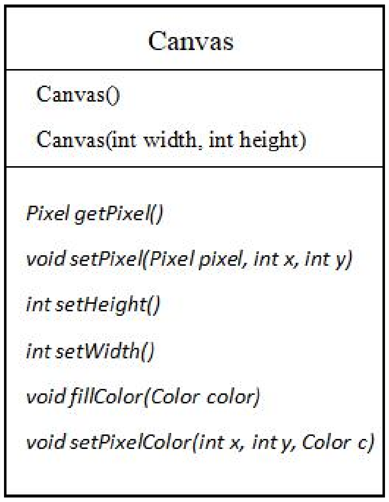

Canvas
Canvas()
Canvas(int width, int height)
Pixel getPixel()
void setPixel(Pixel pixel, int x, int y)
int setHeight()
int setWidth()
void fillColor(Color color)
void setPixelColor(int x, int y, Color c)

Fig. 7. Canvas Class

5) Scene

Scene class stores the data of the scene to be rendered. Generally, there are two kinds of data stored in scene class: light source and primitive. Scene class provides some interfaces for renderer to access those data. When amount of light source and primitive are massive, it will causing the performance problem. To prevent this situation, scene manager is also needed to sort resources like sort opacity object by depth to make them fail the depth test.

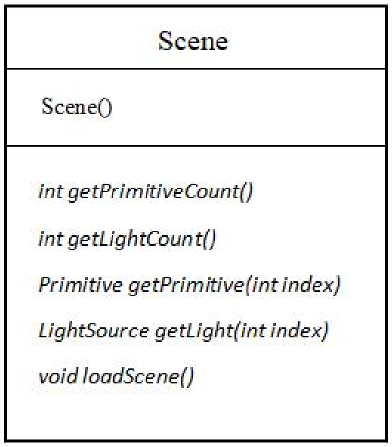


Fig. 8. Scene Class

6) LightSource

LightSource is an abstract class which represents all kinds of light source in scene and provides the pure virtual functions. Different light source inherited from LightSource class just need implement the virtual functions. For ray tracing renderer, the light source should have two member variables, light position for calculating shadow and direction for computing diffuse light and specular light. In this article, to simplify the renderer, only point light and parallel light are implemented.

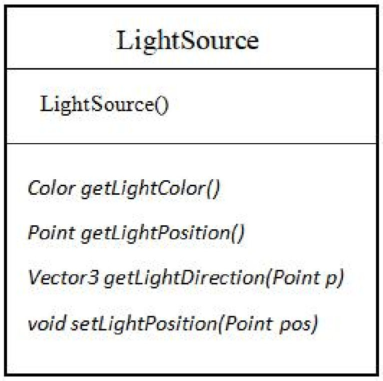


Fig. 9. LightSource Class

7) Material

Material class is used to modify the material of the primitive. Material class stores the important data of the material which can be used by shader. Also, it stores the diffuse reflection coefficient, specular reflection coefficient, refraction coefficient and refractive index. The first three coefficient introduced above describe the energy distribution of diffuse reflection, specular reflection and refraction which add up to one. When recursively tracing the ray of light, the final output will add each light based on this coefficient as weight. And if the specular reflection coefficient or refraction coefficient is zero which means specular reflection or refraction won't occur on this surface, the renderer won't generate relevant light to improve the performance.

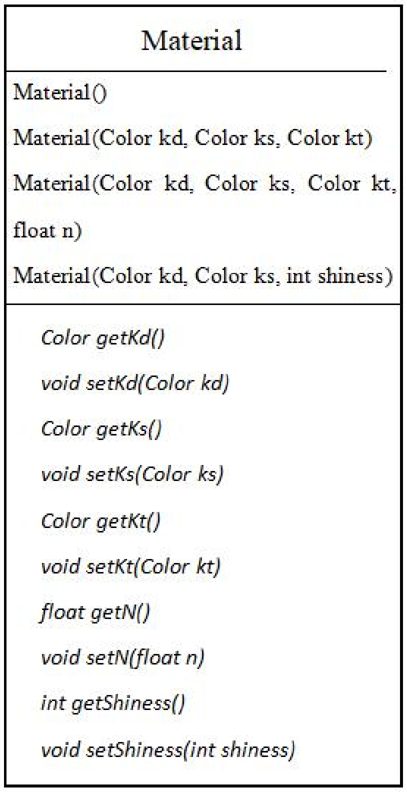


Fig. 10. Material Class

8) *AABB*

AABB(Axis-Aligned Bounding Box) class represent the axis-aligned cube, which completely encircles the primitive. When performing intersection test, the ray of light will run intersection test with AABB first. If the ray not intersect with AABB, the intersection test will stop. Otherwise, the ray will do the test with primitive. Because the AABB is axis-aligned box, it is easy to perform intersection test. By doing this, renderer can greatly reduces the complexity of intersection test and provides a better performance.

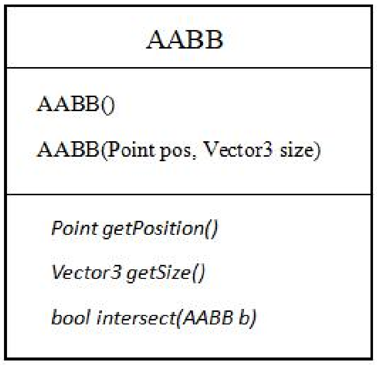
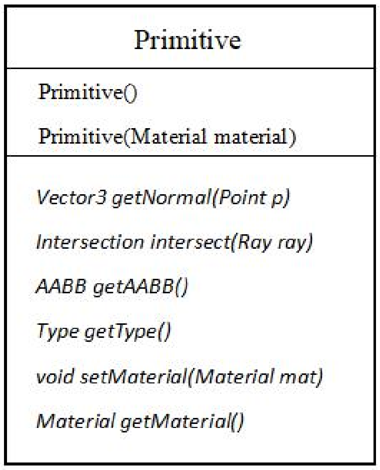


Fig. 11. AABB Class

9) *Primitive*

Primitive class is an abstract class which is the base class of all renderable object. An object must inherit from primitive class and implement its pure virtual class to be rendered. The function in primitive class includes getting normal of a vertex in primitive, getting and setting the material of a primitive, getting the intersection point with given ray. The normal of the vertex is used to define the front and back of the primitive and calculate the reflection and refraction light. The intersection point is one of the most important parameter used by ray tracing renderer. Renderer will choose the closest intersection point and calculate the diffuse light and generate specular reflection and refraction ray of light. The primitive types used by this article are plane, sphere, cube and triangle



Primitive
Primitive()
Primitive(Material material)
Vector3 getNormal(Point p)
Intersection intersect(Ray ray)
AABB getAABB()
Type getType()
void setMaterial(Material mat)
Material getMaterial()

Fig. 12. Primitive Class

.

10) *Renderer*

Renderer class is an abstract class which contains the pointers to scene, camera and canvas objects. This class provides the general interfaces such as render() function. The actual ray tracing renderer will implement this function which is used to generate the original ray of light based on the connection between the camera position and pixel in canvas. The original ray of light intersects with primitive in the scene and recursively generate more ray of light based on diffuse reflection coefficient, specular reflection coefficient and refraction coefficient recorded by material of the primitive. When the this generation reaches the maximum depth setting in the renderer, the color information of all lights will be added up based on the weight of each light carrying out the final output. The process will be done for each pixel in canvas generating the final image as ray tracing renderer's output.

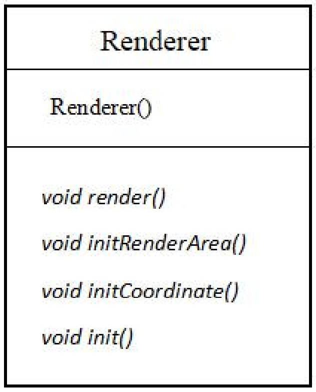


Fig. 13. Renderer Class

B. *Process Design*

The RayTracingRenderer class is the most significant class in ray tracing algorithm introduced by this article. This class provides almost all method required by algorithm. Other class like scene, camera and canvas provide data abstraction and packaging. By using abstraction and inheritance, the ray tracing renderer introduced by this article significantly increasing its maintainability and extensibility. The RayTracingRenderer is performed as follow.

First of all, the init() function of RayTracingRenderer will be executed for initialize the renderer. The initialization includes loading the data into the scene, setting the position and rotation of the camera, setting the size of the canvas and etc. The whole initialization will be divided into some small individual functions such as LoadScene() and InitRenderArea().

After initialization, RayTracingRenderer will generate the original ray of lights rely on resolution of canvas and position of camera. Each ray of light will be represented by the instance of Ray class. Then the RayTracingRenderer will traversal all AABB in the scene for now. World partitioning may be added in further work. For each light which intersects with AABB will progress the further intersection test with primitive. By doing this, all of the intersection points between origin lights and primitives will be worked out and stored in array.

For each intersection point, RayTracingRenderer will traversal light source array and call the getDirection() method. Also the getNormal() method of Primitive class will be called to getting the normal information at intersection point. In same way, the material which contains the coefficients of reflection and refraction will be get by calling the getMaterial() method. Based on all information above, the diffuse color will be figured out using Phone lighting model[6] at each intersection point. After this, the diffuse color will multiple the coefficient of diffuse reflection as a part of final output.

According to the material data at intersection point, the reflection and refraction ray may be generated. The direction information will be calculated based on the incident direction and normal at intersection point following the relevant laws. And the origin of generated ray is the intersection point.

Each generated ray will recursively progress the same calculation as the original light figuring out the diffuse light and generating reflection and refraction light until reach maximum recursive depth. Then, the RayTracingRenderer will backtrack the recursion and add color of reflection and refraction ray of light based on its weight. When backtracking to the original light, the whole process is over, the RayTracingRenderer will record current color as final color.

All original lights will perform the same calculation and call the setPixelColor() method provided by Canvas class. The canvas will figure out the final image after traversal all original light.

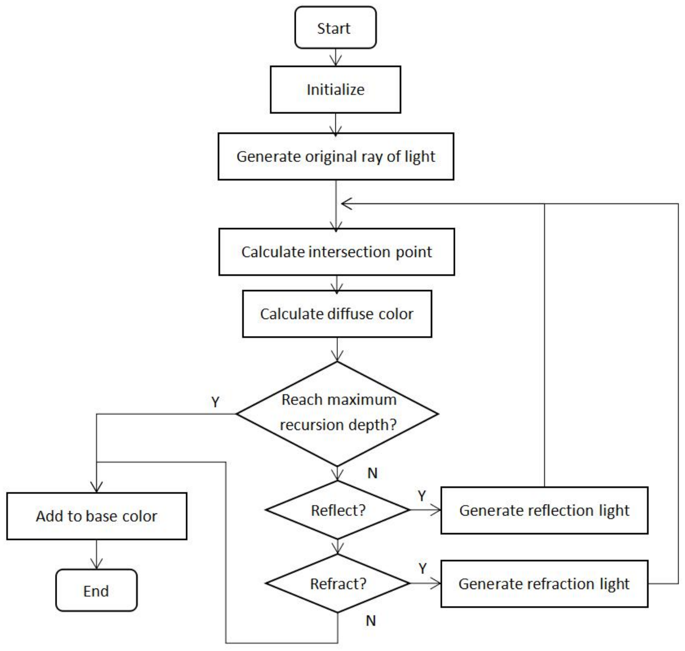

Start
Initialize
Generate original ray of light
Calculate intersection point
Calculate diffuse color
Reach maximum recursion depth?
Y
N
Reflect?
Y
Generate reflection light
Refract?
Y
Generate refraction light
N
Add to base color
End

Fig. 14. General Process

VII. Conclusion

The Ray tracing technique is still a immature technique for game industry. To bring it into practice, lots of optimisation need to be done. In further work, this article will implement the actual renderer and doing some optimisation.

REFERENCES

[1] Cabral, B., Olano, M., & Nemec, P. (1999). *Reflection space image based rendering.* (pp.165-170).

[2] Schulz, N. . The rendering technology of ryse. *Crytek Com.*

[3] Glassner, A. S. (1989). *An introduction to ray tracing.* Morgan Kaufmann Pub. .

[4] Purcell, T., Buck, I., Mark, W., & Hanrahan, P. (2002). Ray tracing on programmable graphics hardware. *ACM Transactions On Graphics*, 21(3). doi: 10.1145/566654.566640

[5] Chiarotti, G. (1973). Optical properties of solids. *Optica Acta International Journal of Optics*, 14(4), 436-436.

[6] Phong, B. (1975). Illumination for computer generated pictures. *Communications Of The ACM*, 18(6), 311-317. doi: 10.1145/360825.360839