CS7332 Computer Graphics

Zhiwei Zhao 022370910020

Project

Illumination models:

$$I_\lambda = I_{a\lambda}k_a O_{d\lambda} + \sum_{i=1}^{N\_light} \frac{I_{d\lambda}k_d O_{d\lambda}(l \cdot n) + I_{s\lambda}k_s O_{s\lambda}(r \cdot v)^{sn}}{a + bd_i + cd_i{}^2}.$$

In this application, the rendered color is not calculated in each RGB channel individually, but is employed as vector multiplication, so the equation turns into a coded form as:

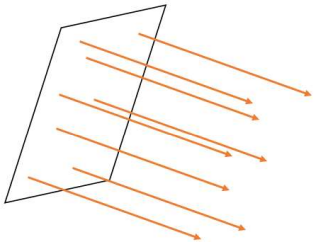
$$R(x) = \sum_{i=1}^{N\_light} R_{i,a}k_{i,a}R_{x,d} + \frac{R_{i,d}k_{i,d}R_{x,d}(l_{i,x} \cdot n_x) + R_{i,s}k_{i,s}R_{x,s}(r_{i,x} \cdot v_x)^{sn_x}}{a + bd_{i,x} + cd_{i,x}{}^2},$$

where x stands for which position of which fragment is being rendered. The total number of N_light light sources are considered. Any light source i can let its ambient, diffuse, and specular colors onto objects be different as $R_{i,a}$, $R_{i,d}$, $R_{i,s}$, but normally they can be the same. Each of these kinds can have different strength as $k_{i,a}$, $k_{i,d}$, $k_{i,s}$, where in usual case the ambient strength $k_{i,a}$ is much smaller than others, and the specular one $k_{i,s}$ may the largest. Similarly, the object color can perform differently for different kinds of lights, when $R_{x,d}$ and $R_{x,d}$ often stands for different texture being used as to behave like a dual-layer surface for some visual interest. sn_x stands for the local shininess, the higher of which the more concentrated specular high lights would be. $l_{i,x} \cdot n_x$ calculates the diffuse shading, and $r_{i,x} \cdot v_x$ calculates that of specular light. If replacing with the halfway vector, it turns form Phone shading model to Blinn-Phong model for saving the calculation.

Different types of light sources

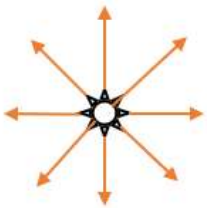
Apart of the basic formula, light sources can have different types, and in this project three are implemented.

Direct Light:



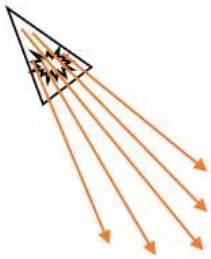
Light directions are parallel within some region from a plane, so its light directions are identical for fragments on all objects if illuminated. Also, no distance attenuation for this type as the light intensity is constrained parallelly in the space.

Point Light:



Light directions are originated from a point. A common type that is good to show object's specular high light. Attenuation must be considered as the intensity decrease into every solid angle reaching farther space.

Spotlight:



Similar to point light, but it doesn't go to every space direction. It is constrained within certain solid angle toward some direction. Attenuation needs to be considered, and at the edge of the illuminated region, there should be a soft transition from light to darkness.

Determination of object geometries

For simple geometry, the vertex coordinates, normal directions and texture coordinates can be self-drawn one by one. While drawing more complex objects, it is good to be assisted by some software like Blender, and then read the exported .obj file. In OpenGL, it thus needs a function to convert .obj type to a float array for reading. Besides, it is better to use smoothed object for curvature faces in spheres as well as side of cones and cylinders. Meaning that vertex of the same coordinate should have the same normal direction, so that the fragment of the surface can interpolate the triangle normals.

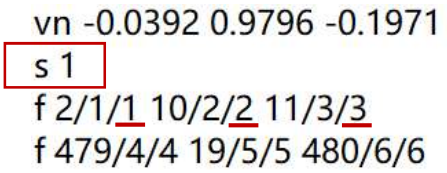


Fig. Active smoothness in .obj file, so face normals can have different values.

Use of texture

The texture simply provides specified color upon a fragment, it looks good for regulated plane to be filled with some pictures and also more realistic for any object with targeted material outlook. All needed stuff are texture coordinates and corresponding picture. Especially, texture could be used for material's hetero sensitivity to diffuse and specular lights as aforementioned. Also, texture can be used to assist shadow mapping which would be introduced later.

Dealing with shadows

Shadow mapping:

Usually, calculating whether some fragment is under the shadow demands determining if it is in the shadow volume of any object from certain light sources. However, when the object turns complex and the number increases, this process

would demand higher computational time so that not friendly to the hardware. Instead, an efficient way of drawing shadows is so-called shadow mapping or depth mapping. Namely, the program will first change its camera position and orientation the same as the light source, and then calculates only the Z-buffer that fully determined from merely the geometry. The stored information in the framebuffer would not be drawn but translated to a depth texture in the view port coordinate. This would be conducted for every light source. Then later in the normal rendering procedure, every fragment position would be again mapped onto the light view and whose' depth is compared against pre-stored depth texture in the same coordinate. If the current depth is larger than pre-stored one, then this position is now in the shadow of that light and denoted by some flag. For fragment component in the shadow, diffusive and specular lights can be neglected, while the ambient light can be reduced but definitely remains some degree.

Soft shadow:

In real physical situation, due to non-regular light diffuse and possible refraction, the shadow should not appear with clear edge, but with a soft transition. Also in the computer program, due to insufficient resolution, first order interpolation, and numerical roundoff error, the edge of the shadow also possibly appears zig-zag shape. To tackle this issue, the local shadow value s (0 for not in shadow and 1 for in shadow) can usually be filtered with some kernel functions, or simpler, averaged with the neighbor. So, the shadow value $\tilde{s}$ turns from integer to fraction, and one can mix the rendered color accordingly:

$$(1-\tilde{s})(amb+diff+spec)+\tilde{s}(reducefactor\cdot amb+0+0).$$

Light though transparent object:

Additional treatment shall be paid with the interaction of shadow and transparent objects. Drawing transparent objects itself can be helped a lot with Z-buffer, but when considering their shadows there are usually two aspects of additional treatments. The first is, how the light color change passing through semitransparent stuff. And the order of shadows by solid objects and transparent

ones. In this project, the treatments are as following.

Suppose the light before an object with transparency a (0 stands for fully transparent and 1 stands for fully opaque) is of the color and strength of $light\_i$, the transparent object has the color of $color\_trans$, so the passed light is with the color and strength as $light\_o$, which equals

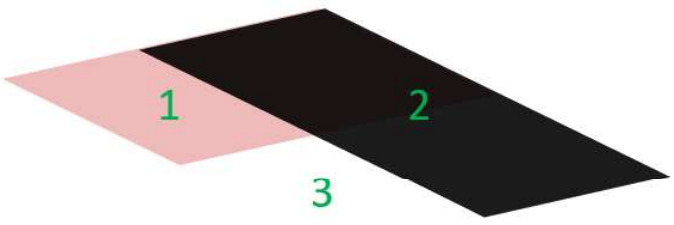
$$(1-a)\cdot(lig\_i\cdot(1-a)+trans\_color\cdot a),$$

when $a=0$, $ligh\_o=light\_i$ as the fully transparent object does not affect anything. While $a=1$, $light\_o=0$, just meaning no light can pass through, this part is just under pre-defined shadow.

For the order of evaluating transparent objects and shadows, it is, firstly determining whether current fragmentation location is under any shadow of opaque object or under no transparent object from certain light source just like aforementioned. If it is, no bother to take transparent objects into account and everything is like before. Otherwise, first determining what is the incident light color and strength from formulas of $light\_o$. Note that $light\_i$ can pass other transparent objects, being $light\_o$ of previous layer. And by this color, calculates any ambient, diffuse and specular light components.

Soft shadow with transparent lighting:

Now, one possible situation is that some shadow edges is at the intersection of both opaque shadows, transparent shadows and direct exposure. The way of dealing this is a two-layer mixing.



As illustrated in the figure, around the cross-intersection point, we have both $\tilde{s}$ and $\widetilde{s_t}$, meaning how much it is under the opaque shadow and how much is under the transparent shadow, accordingly. Fraction of area 2 can be expressed as $\tilde{s}$, fractions of 1 and 3 are each (1-$\tilde{s}$)$\widetilde{s_t}$ and (1-$\tilde{s}$)(1-$\widetilde{s_t}$). Thus, the local light color can

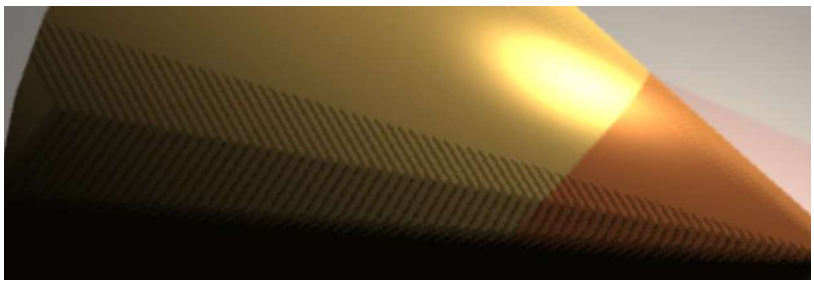
be:

$$\tilde{s}(reducefactor \cdot amb) + (1 - \tilde{s})(1 - \widetilde{s_t})(amb + diff + spec)$$
$$+ \tilde{s}(1 - \widetilde{s_t})\left(amb + diff + spec|_{light\_o}\right).$$

Insufficiency of shadow mapping and treatments:

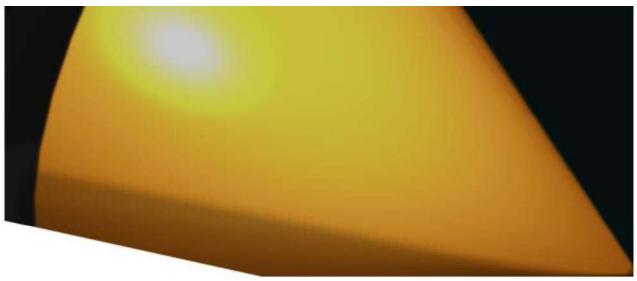
Some issues may arise for the method of shadow mapping, and corresponding solutions are provided in this project.

1. Shadow acne.



It looks something like a Moire pattern and is actually caused by matching the depth in the texture map and actual rendering scene. Mostly this is due to roundoff error of float precision, and increasing the resolution can help but not efficiently. In some tough location where the light view to the surface is so inclined, such acne is most hard to be eliminated.

Solution: Adding a depth bias, and it can be dynamically adjusted by the inclined angle of view direction and fragment face. Note this value can not be too big otherwise shadow would shift incorrectly.



2. Shadow texture range.

Firstly, shadows texture should be mapped with perspective view. Possibly, some shadow relations are cut off in light view, but appear in camera view. And to the farther space, shadow can repeat abnormally like a repeated texture.

Solution: Increase the perspective angle to count most of shadow relations in

light view and define no repeat of shadow texture where father space sets shadow depth the smallest.

Algorithm: Code structure

Plat form: Linux (Ubuntu 22.04)

Language: c++

Include library: std, GLEW, GLFW, glm (for math calculation), SOIL (read image into texture).

Compiling: std c++11, cmake

Defined functions

Obj file reader: read .obj file into vertices, normal, texture coordinates array.

Callback functions: control the camera.

Shaders (compiled in GPU): Four pairs of shaders. One for drawing lights (point light), one for rendering the shadow map (no color drawing), one for drawing objects with the texture, and one for objects with uniform color and transparency. Required GLSL version 4.0+, as I used a sampler array and dynamic index.

Structures

/*--*/

set global parameters;

read provided .obj files;

set for GLFW and create window;

set the callback functions;

compile all shaders;

self-define some rectangles and tetrahedron;

create buffer: VAO, VBO, bind and set the vertex array;

configure the depth framebuffer and link to depth textures, for all lights and additionally transparent objects;

build texture from provided images, one used for diffusive light is an AI generated painting, the other used for specular light is human drawn;

loop according to camera movements

{

calculate depth into depth buffer and generate shadow map, not draw to screen and stored in depth textures (for each light and separately transparent objects);

set uniform parameters for the phone shaders, now draw objects with image textures (polyhedrons);

change the shader to draw lights (just pure light color close to white);

change the shader to draw other objects (ground, sphere, cone, cylinder);

}

Done;

/*----------*/

In the phone fragment shaders, it looks like:

struct Material;

struct DirLight;

struct PointLight;

struct SpotLight;

To define amb, diff, spec light colors, strength, light position and direction, attenuation factors, spotlight angles, etc.

in vertex position;

in light positions;

out rendered color;

uniform settings;

All light and material structures, shadow map textures, transparency, transparent object colors, etc.

Add all kinds of light results:

CalcDirLight();

CalcPointLight();

CalcSpotLight();

ShadowCalc();

All kinds of light are made up of ambient, diffuse, and specular components. Shadows are considered both for $\tilde{s}$ and $\widetilde{s_t}$, and mixed together to being soft, fully in opaque shadow has only reduced ambient

lights, in exposure has normal light components, and in transparent shadows have decreased light strength due to the object transparency.

Detailed Parameter settings

Number of light sources: two point lights

Constant coefficient:

light color: (1.0, 0.9, 0.8)

ambient strength: 0.125

diffuse strength: 1.25

specular strength: 2.375

Attenuation factors: $a = 1$, $b = 0.09$, $c = 0.032$

Shininess of the material specular behavior, $sn = 32$

no-texture object color: (255, 227, 132)/255

ground color: (0.4, 0.4, 0.4)

transparent object color: (1, 0, 0) pure red

its transparency: 0.2

Tips：

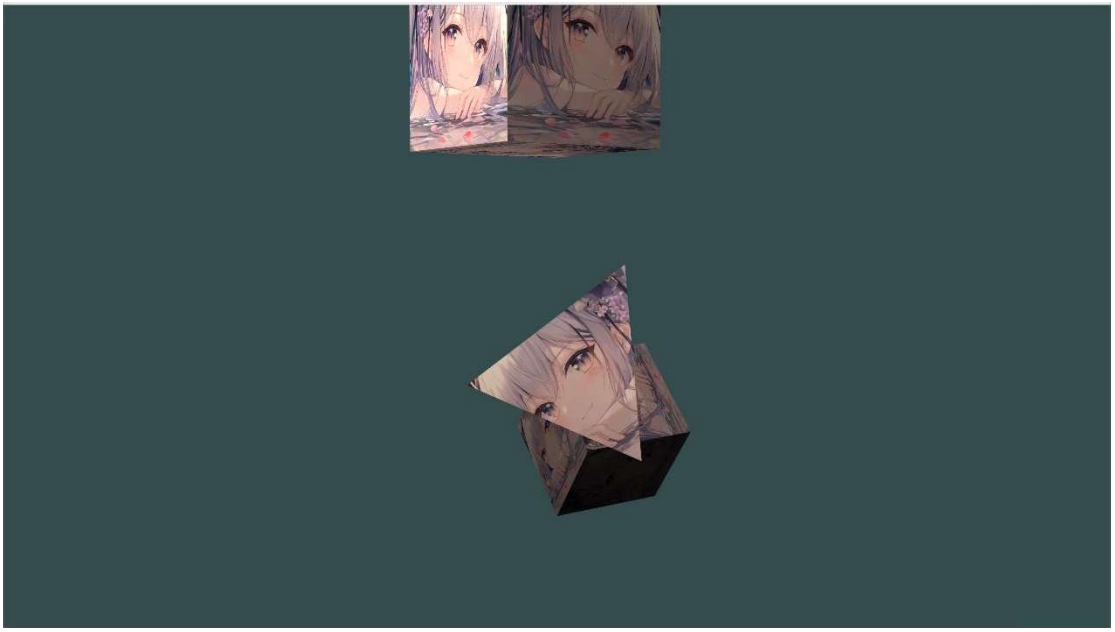
GLFW window: 3328 × 1872, ok in 4K screen, may reduce it with smaller resolution.

The program works hard in core-GPU if laptop only has it but can work well in my PC RTX2080Ti.

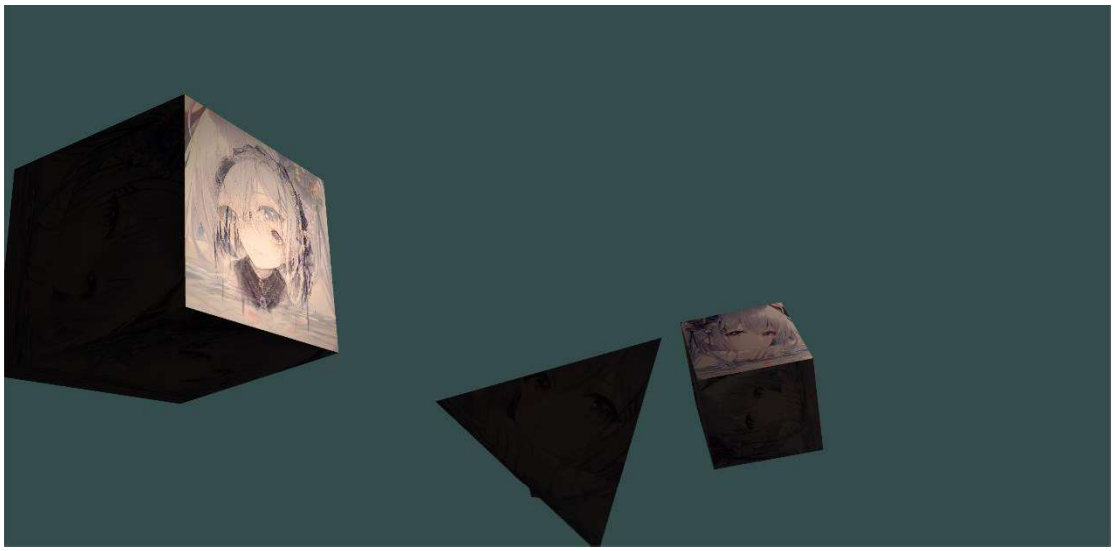
Uses oversampling 2×2 for anti-aliasing, no need for gamma-correction as not simulating a real scene.

Results

Only ambient and diffuse, the dark side shows an ambient strength as 0.125:

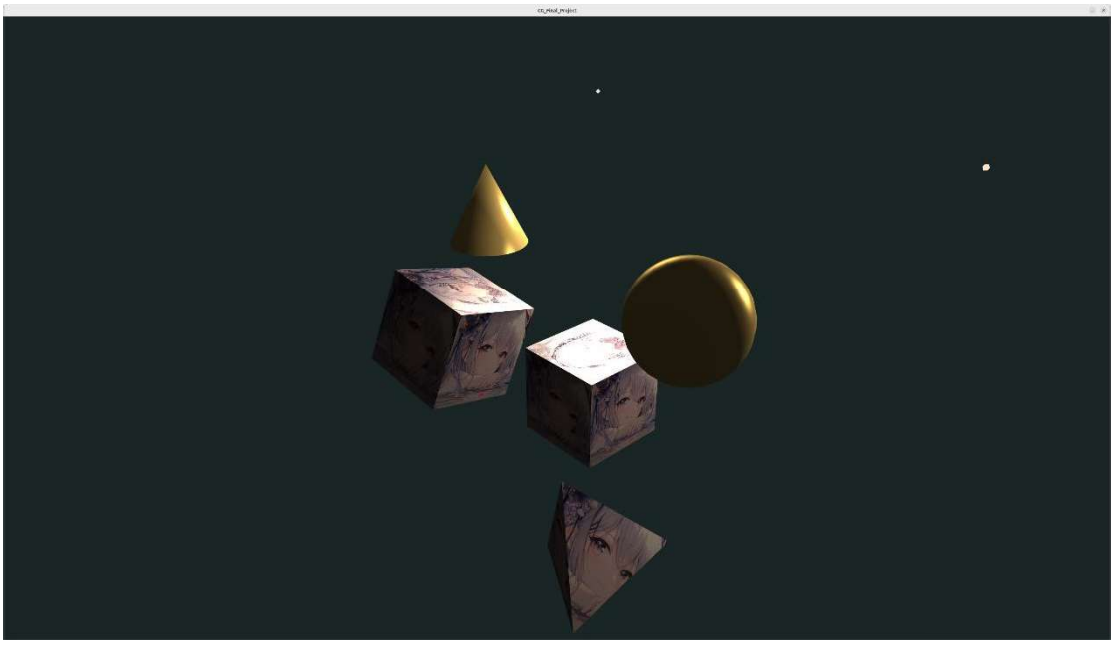


Specular lights use another texture:

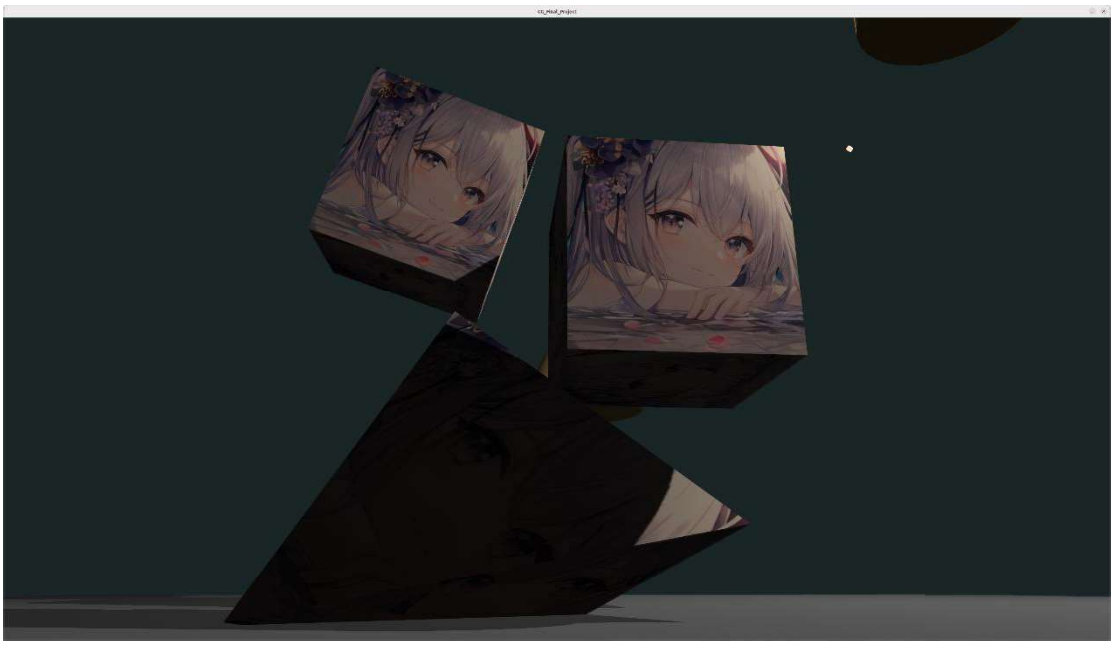


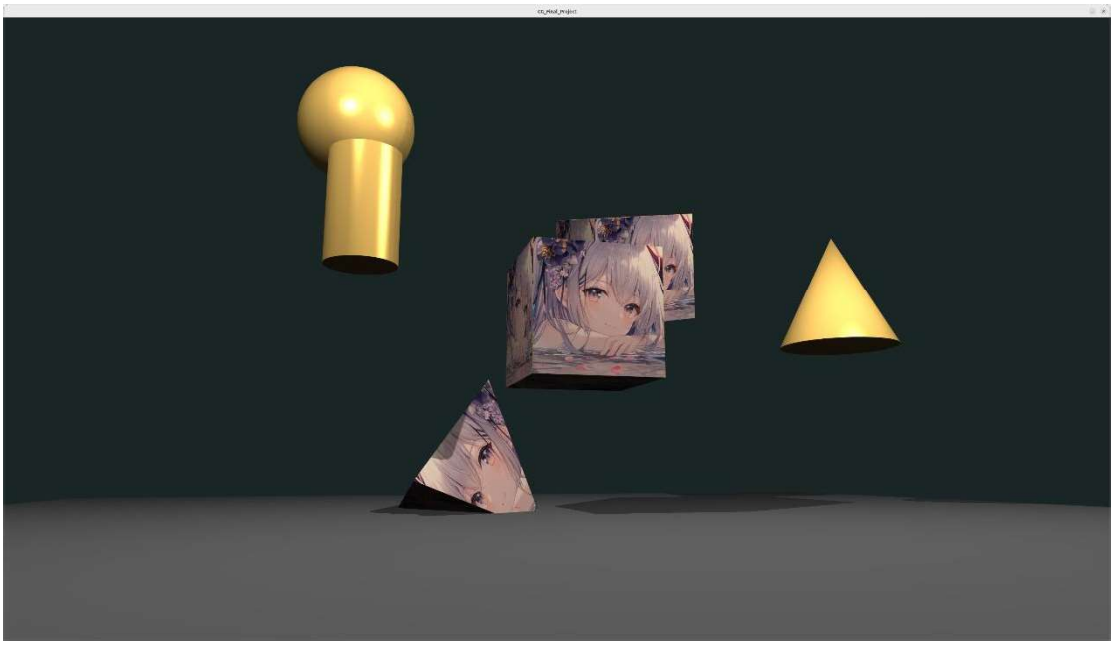
Some angle view:

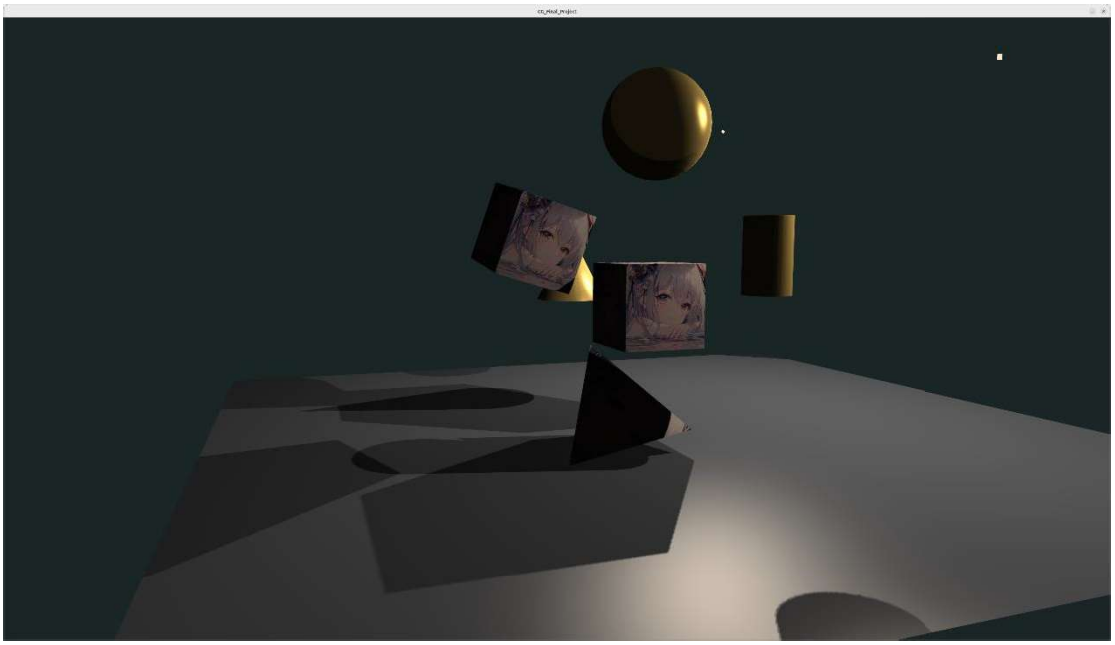
Adding other objects:

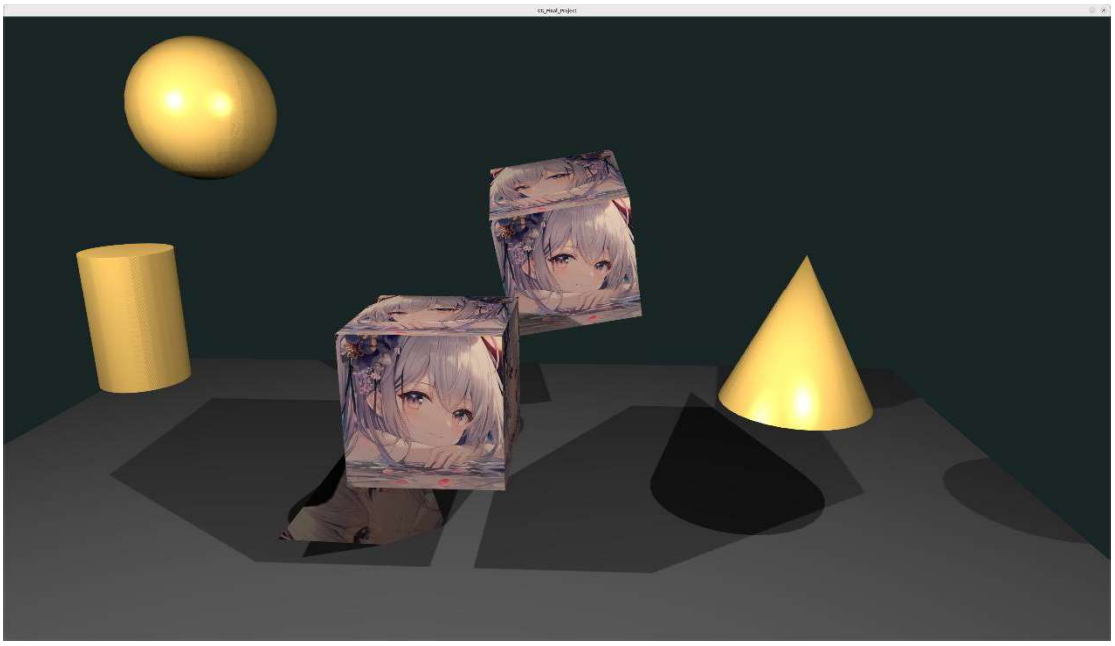


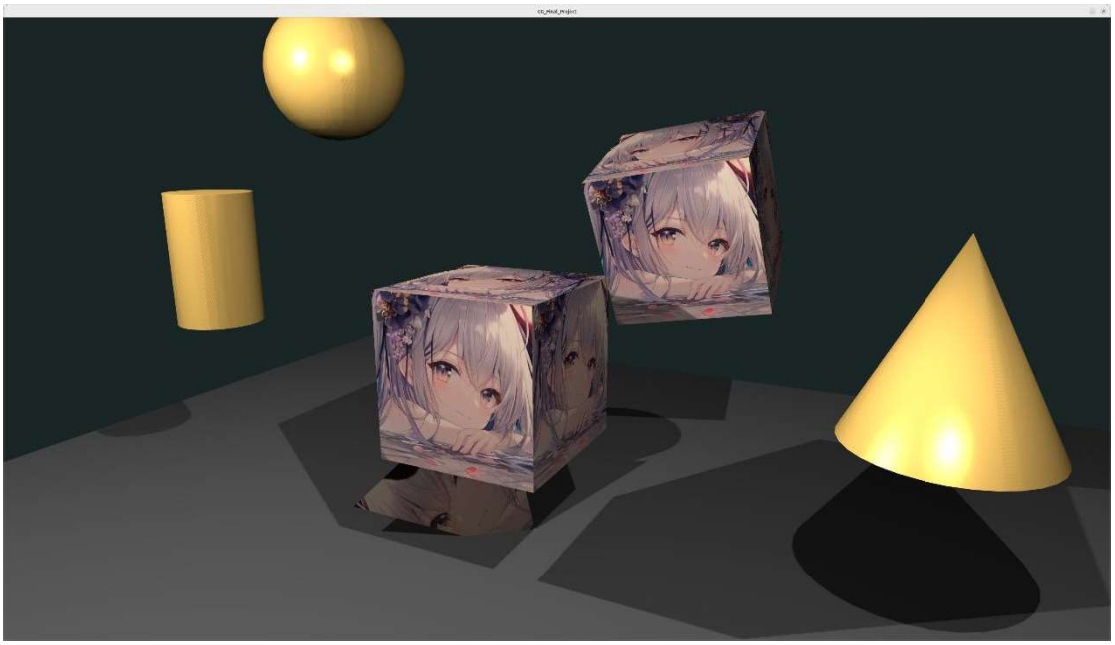
With shadows:







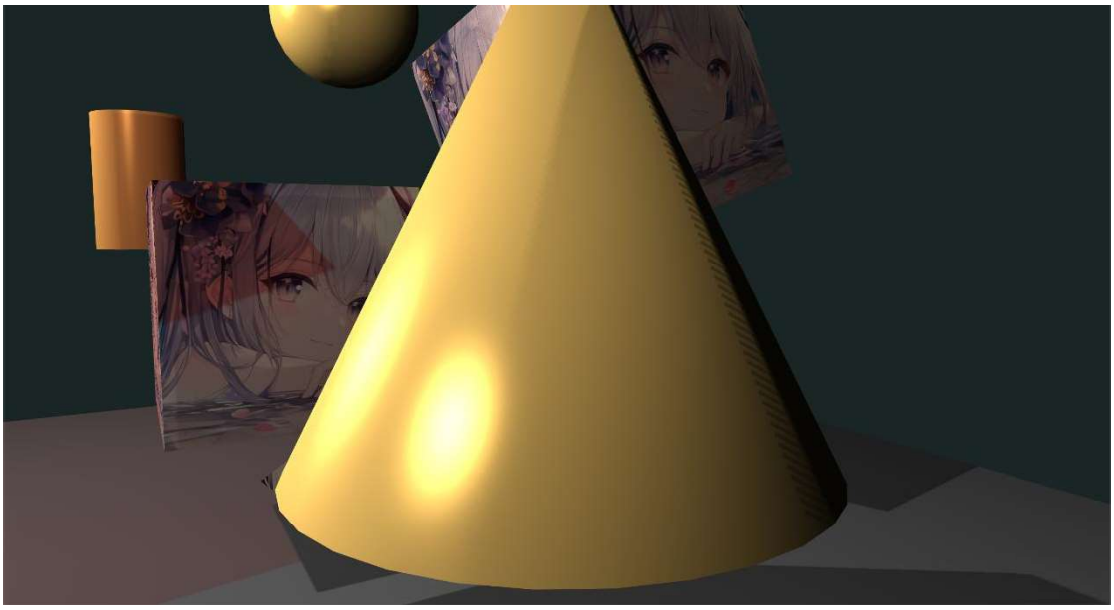




Dealing with the shadow acne:

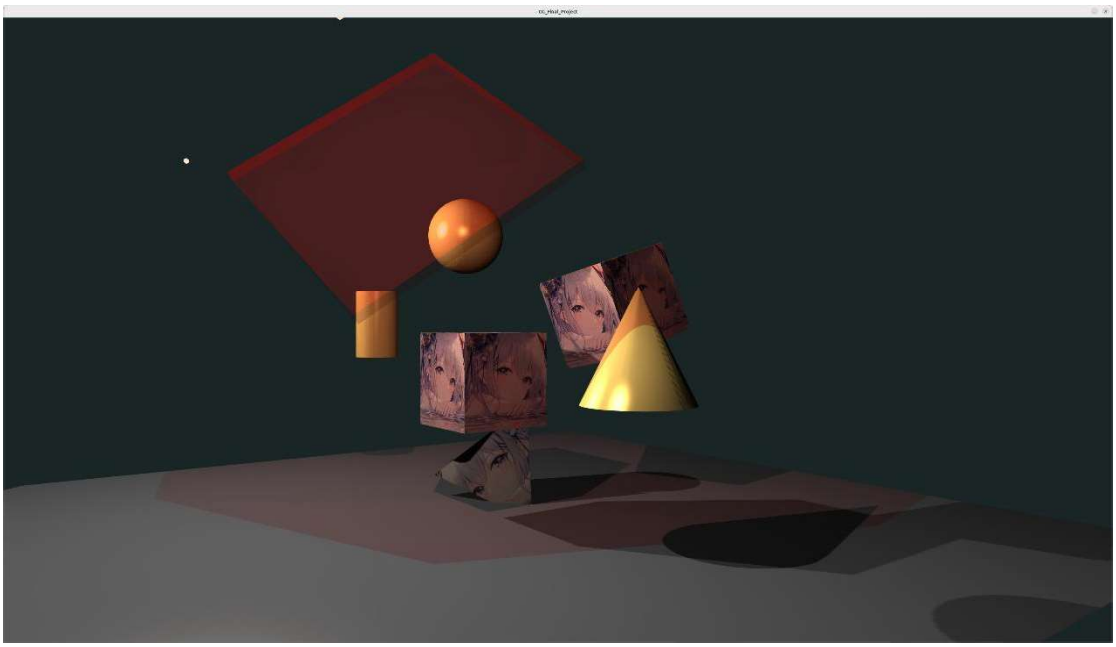
worse:

better:



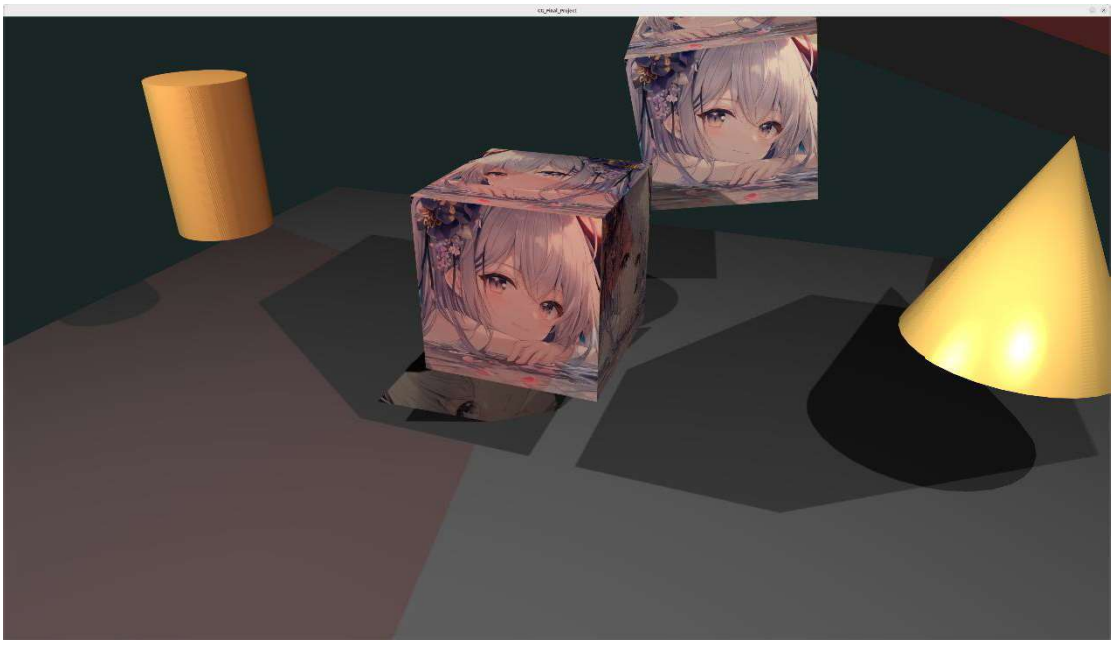
Global shadow view:

worse:

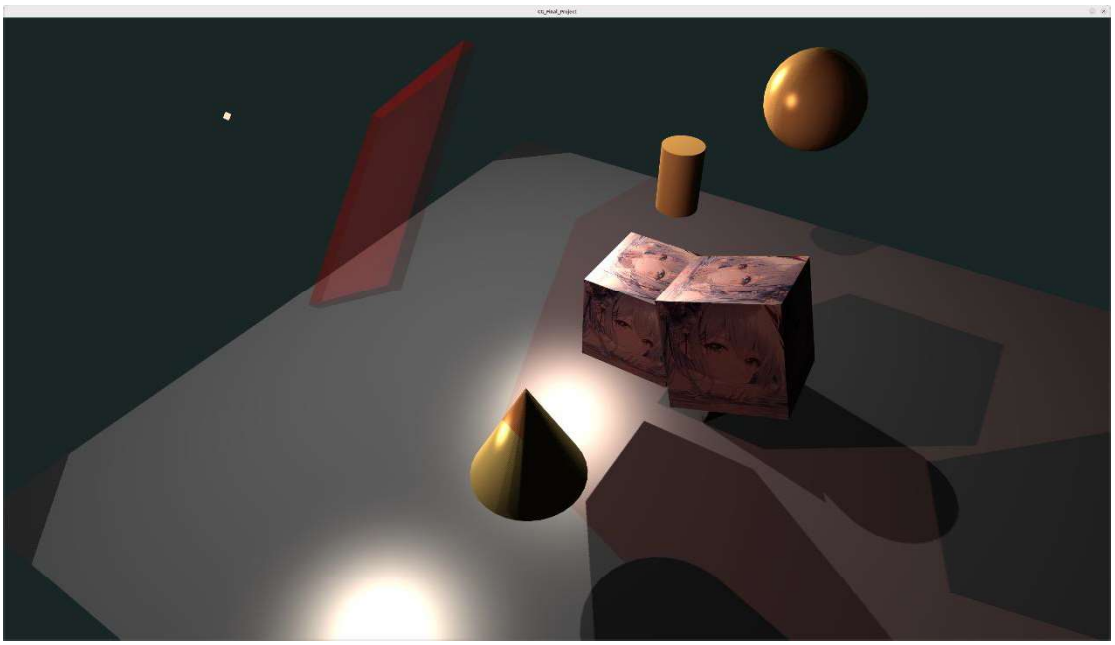


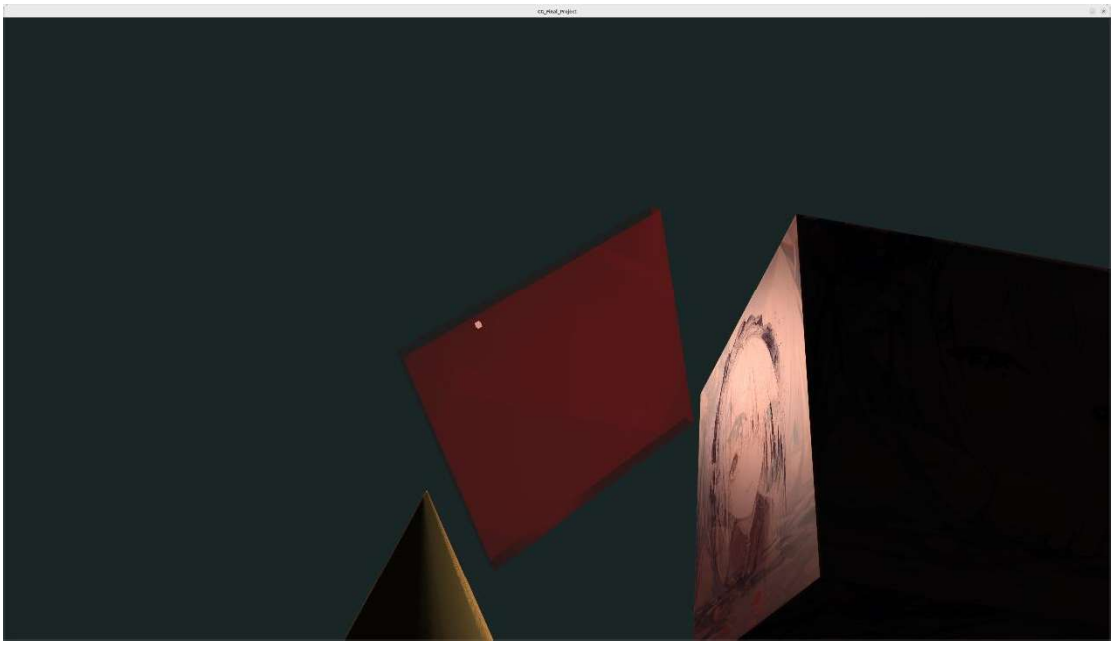
better:

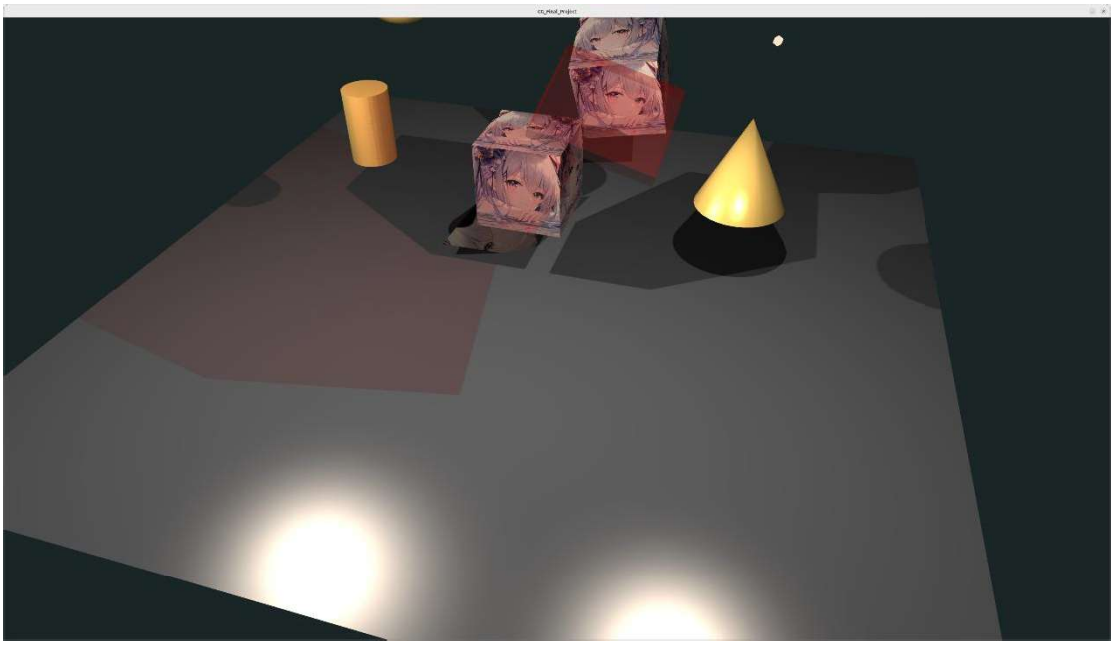
Change the transparent object size:



Some others, shadow, transparency, high light:







Other screen shots and a video can be found in the attached file, providing more aspects.

Attached codes

source code project.cpp and CmakeList.txt settings, enter build directory and *cmake ../*, then *make*. Some paths of headers and libs may need to be modified according to user's situation.

Please make the executable file together with images and .obj files.