Shading and Shadows

Computer Graphics II Final Project

CHENYANG ZHANG, ShanghaiTech University, LDRFans
ZIQI GAO, ShanghaiTech University, LDRFans

This is the final project for CS271 Computer Graphics, ShanghaiTech University. You can see all the codes in the our GitHub Repository: https://github.com/LDRfans/Shading-and-Shadows.

ACM Reference Format:
Chenyang Zhang and Ziqi Gao. 2021. Shading and Shadows: Computer Graphics II Final Project. 1, 1 (June 2021), 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Physically-correct visual effect in respect to lighting is of crucial importance to bring out 3D appearance and achieve realism in a rendered 2D image. Effects needed for realism include shading, shadows, reflections, transparency, inter-reflections, complex illumination, etc. In this project, different approaches for the first two effects (shading and shadows) are implemented and compared respectively based on a series of designed experiments.

2 METHOD

2.1 Shading

Assuming a light source with two lights and an environmental ambient illumination is given, we use Phong illumination model, which describes the interaction between light and surface, to approximate the physical light reflection.

2.1.1 Phong Illumination Model. As equation 1 shows, the Phong illumination model consists of four parts: the material color K_e, the ambient reflection L_a, the diffuse reflection L_d, and the specular reflection L_s.

$$L = K_e + L_a + L_d + L_s \tag{1}$$

Ambient Reflection. The ambient reflection describes the reflection of the ambient light, whose intensity is assumed equal in all directions. Given the ambient reflection coefficient k_a and the ambient light intensity I_a, the ambient reflection to one point is

$$L_a = k_a I_a \tag{2}$$

Authors' addresses: Chenyang Zhang, ShanghaiTech University, LDRFans, zhangchy1@shanghaitech.edu.cn; Ziqi Gao, ShanghaiTech University, LDRFans, gaozq@shanghaitech.edu.cn.

XXXX-XXXX/2021/6-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

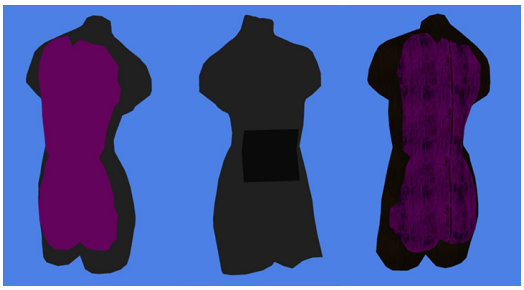


Fig. 1. Venus with ambient illumination only.

Diffuse Reflection. The diffuse reflection has equal intensity in all directions, so it is not dependent on the direction of viewer, but dependent on the direction of the incoming light and the surface material. The diffuse reflection can be modeled as

$$L_d = k_d \sum_i f(d_i) I_{li} (\mathbf{N} \cdot \mathbf{L}_i)_+, \tag{3}$$

where k_d is the diffuse reflection coefficient, I_l is the intensity of the light source, $\mathbf{N}$ is the surface normal and $\mathbf{L}$ is the direction of the light source. Since the diffuse light cannot penetrate surface, only positive $\mathbf{N}\cdot\mathbf{L}_i$ is preserved by the $()_+$ operator. Note that the intensity of a point light source drops off with distance, the attenuation factor $f(d)$ is defined to describe this drop-off by

$$f(d) = min(1, \frac{1}{a + bd + cd^2}),$$

where a, b, c are supplied by users.

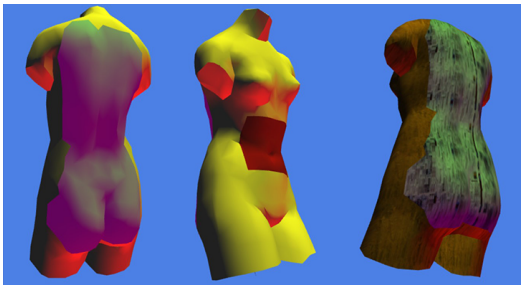


Fig. 2. Venus with ambient illumination and diffuse illumination.

Specular Reflection. Specular reflection describes the highlight on the object. Different from the diffuse reflection, the specular light also depends on the viewing direction $\mathbf{V}$ and the color often solely depends on the light source. The specular reflection can be modeled

as

$$L_s = k_s \sum_i f(d_i) I_{li} (\mathbf{V} \cdot \mathbf{R}_i)_+^{n_s} \tag{4}$$

with the viewing direction **V** and reflection direction **R**.
In this project, we choose a simplified Phong Illumination model, Blinn-Phong Illumination model, to speed up the specular reflection computing process. Here we replace **V** and **R** with the surface normal **N** and the half-vector $\mathbf{H} = \frac{\mathbf{R}-\mathbf{L}}{2}$ indicating the intermediate vector of the incident light direction **L** and the viewpoint direction **V**. The equation of the Blinn-Phong Illumination model is shown below.

$$L_s = k_s \sum_i f(d_i) I_{li} (\mathbf{N} \cdot \mathbf{H}_i)_+^{n_s} \tag{5}$$

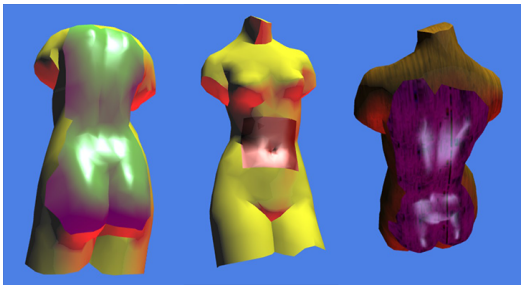


Fig. 3. Venus with ambient illumination, diffuse illumination and specular illumination.

Phong Illumination Equation. So far, by combining equation 1 2 3 4, we can derive the *Phong Illumination Model*

$$I = K_e + k_a I_a + \sum_i f(d_i) I_{li} [k_d (\mathbf{N} \cdot \mathbf{L}_i)_+ + K_s (\mathbf{V} \cdot \mathbf{R}_i)_+^{n_s}], \tag{6}$$

and the *Blinn-Phong Illumination model*

$$I' = K_e + k_a I_a + \sum_i f(d_i) I_{li} [k_d (\mathbf{N} \cdot \mathbf{L}_i)_+ + K_s (\mathbf{N} \cdot \mathbf{H}_i)_+^{n_s}]. \tag{7}$$

2.1.2 Shading Methods. 3D mesh objects are usually represented by polygons with different normal directions. However, as Fig 4 shows, the color of pixels in one polygon is the same for they share the same normal direction, leading the normal direction as well as the light intensity change suddenly with the adjacent polygons making a discontinuous light intensity. Shading is a approach to eliminate this visual problem by interpolating light intensity or parameters within each polygon, enabling us view more smooth-looking objects under different light environments. Here we use *Gouraud Shading* and *Phong Shading*.

Gouraud Shading. The main idea of Gouraud Shading is simple: first calculating the average normal of polygon vertex, then calculating the light intensity of each vertex, then do the light intensity bi-linear interpolation for each point in the inner domain of a polygon, using the light intensity value of vertex on that polygon.

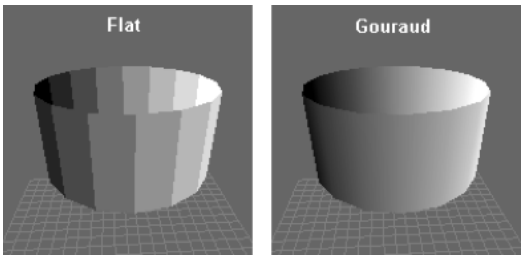


Fig. 4. Flat vs Gouraud Shading.

Algorithm 1 Gouraud Shading

```
let zBuffer
let frameBuffer
for let object of objects do
    for let triangle of object do
        let reflections = computeReflections(triangle)
        for let vector3 of triangle.pixels do
            if vector3.d < zBuffer[vector3.x][vector3.y] then
                zBuffer[vector3.x][vector3.y] = vector3.d
                frameBuffer[vector3.x][vector3.y]
= interpolate(reflections, vector3.x, vector3.y)
```

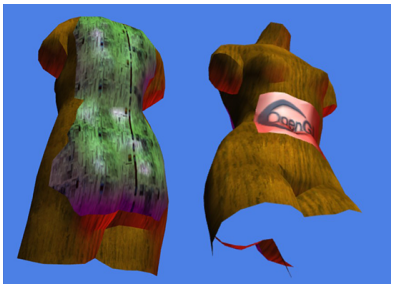


Fig. 5. Venus in Gouraud Shading view.

Phong Shading. Different from the Gouraud Shading method, Phong Shading interpolates the normal of vertex on one polygon. First, calculating the average unit normal vector at the vertex of each polygon, then interpolating the vertex normal to derive the normal vector of each point. Finally, the light intensity of each point can be calculated independently using the Phong Illumination model.

Comparison between Gouraud and Phong. The Gouraud Shading model is simple and fast, uniformly changing the light intensity, while its shading result is acceptable in most cases. However, as fig 7 shows, the simple light intensity interpolation may lose most specular light when the vertex is sparse since the vertex may not in the position where the specular light is highest.

Phong Shading solves this problem by replacing the light intensity interpolation with normal interpolation, preserving normal for each point, so that the highlights can locate inside the polygon though its adjacent vertex's intensity is not high. But this improvement is at the cost of time, since we must interpolate the normal of each point and compute its light intensity.

Algorithm 2 Phong Shading

```
let zBuffer
let frameBuffer
for let object of objects do
    for let triangle of object do
        let normals = getNormals(triangle.vertices)
        for let vector3d of triangle.pixels do
            const  x, y, z  = vector3d
            if z >= zBuffer[x][y] then
                continue
            let normal = interpolate(normals, x, y)
            frameBuffer[x][y] = getIFormula(normal, object)
```

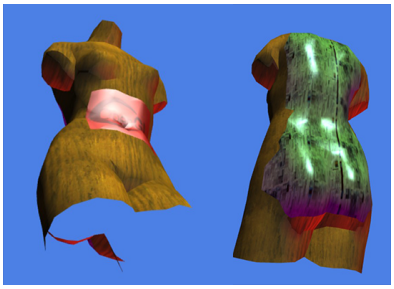


Fig. 6. Venus in Phong Shading view.

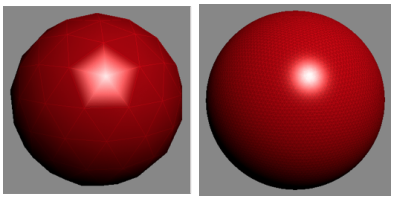


Fig. 7. A potential downside of Gouraud Shading: specular light may seem dim if there is not high enough resolution of triangles.

2.1.3 Texture Mapping. Texture mapping is the process that replacing objects' surface with image details. Basically, it projects an 2D image to the 3D object's surface. We can add texture using the GLSL built-in function *texture2D()*, given the color map and texture coordinate.

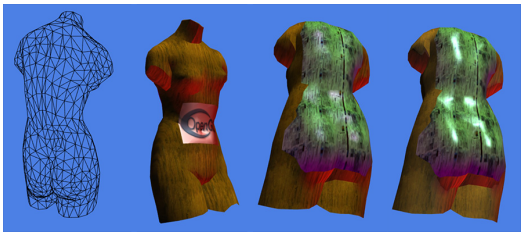


Fig. 8. Venus: the Hidden Line view, Textured Smooth Shaded view, Gouraud Shading view and Phong Shading view.

2.1.4 Animation. We do the vertex animation by controlling each vertex's x position. You can see the video in the report folder.

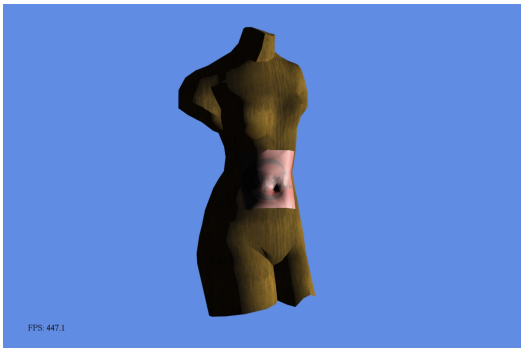


Fig. 9. Video Screenshot.

2.2 Shadows

The reason why shadows are important in rendering is straightforward: it reflects scene lighting, provide depth cue and relative location, etc. In ray tracing, whether a point is in shadow of a light is determined by tracing the secondary shadow rays towards the light source. If the distance of the closest point hit in the ray, then the point is in shadow. Intuitively, the algorithm is done by tracing the ray given by each pixel and see whether the intersecting point is blocked by some other objects in its way to the light. However, in rasterization, secondary shadow rays cannot be found and computed using the algorithm since one object is considered at a time. Yet there are still two kinds of algorithms that can cast shadows in the rasterization's setting, namely shadow mapping and shadow volume, which will be introduced in the following sections.

Fig. 10. An Example of Shadow in a 2D Image.

2.2.1 Shadow Mapping. Shadow mapping involves two passes of rendering. The first step is to render from the point of view of the light and generate a depth map, which is shown in the left-down image of Figure 11. This is quite simple and can be considered as a z-buffer seen from the light. The second step is to render from the camera or eye's point of view. For each point, we can check whether it was in the shadow map by checking whether its depth and the depth that can be looked up from the previous the shadow map. For Figure 11, v_a is not in shadow since $d_a = d'_a$; v_b is in shadow since $d_b < d'_b$.

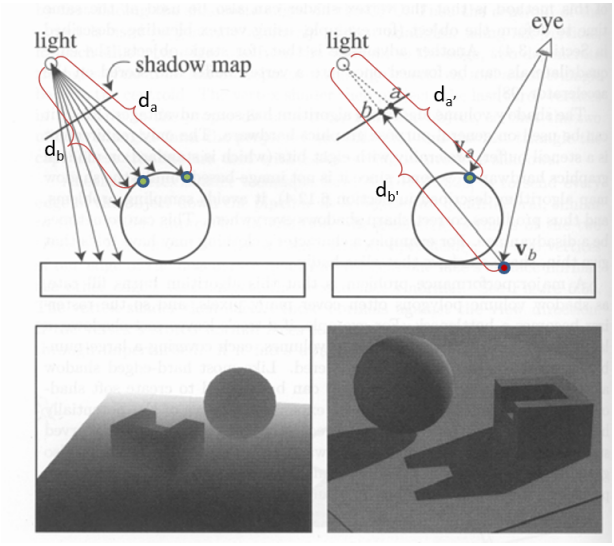


Fig. 11. Illustration of Shadow Map Algorithm.

In practice, the two depth d and $d^{'}$ is not equal even if the corresponding point is not in shadow due to insufficient precision of shadow map. This can be solved by adding a $shadowZoffset$ in comparison, which will be discussed in the experiment part.

The algorithm pipeline of lighting with shadow mapping in shown in Algorithm 3 and a visualization of shadow map is shown in Figure 12. Although hard to notice, one needs to notice that the intensity of color in Figure 12 is not the same, and it reflects the correct depth value of two light in the code of our project.

Algorithm 3 Lighting Outline with Shadow Map

```
procedure LightingWithShadowMap(δ:shadowZOffset)
    Render the ambient light into the color buffer
    for each light source do
        Compute depth d from the light's point of view and render the shadow map M
        Compute the shadow map coordinate and depth d' in texture space and compared it with the fetched depth from the shadow map M
        if |d' − d| < δ then
            Blend diffuse and specular light into the final color
```

2.2.2 Shadow Volume . The Shadow Volume algorithm is the other algorithm for shadow casting, where the volume of space in shadow is represented explicitly as a shadow volume. In theory, shadow volume is the half-space intersection defined by a shadowing object and a light source, which is of infinite extent. In implementation, it is clipped in a certain depth. A naive implementation of Shadow Volume is shown in Algorithm 4, which is quite computationally expensive.

Z-pass is a nicer implementation, where a ray is shoot from the eye to the visible point and a counter is incremented or decremented each time the ray intersect a shadow volume polygon. If the counter

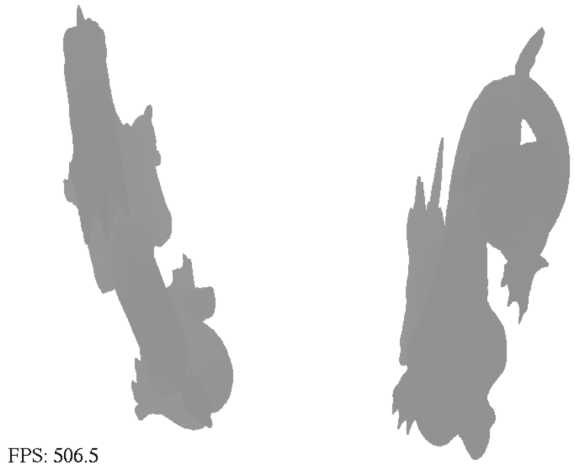


Fig. 12. Visualization of Shadow Map Algorithm.

Algorithm 4 A Naive Shadow Volume Implementation

```
for each light source do
    for each polygon do
        Create a pyramid with point light being apex
        Create a shadow volume according to the pyramid
        Add it to the set of shadow volumes
```

≠ 0, the point is in shadow. That is because the ray does not go in and out for all shadow volumes. An example of this algorithm is shown in Figure 13, where the object is not shadowed since the count is 0.

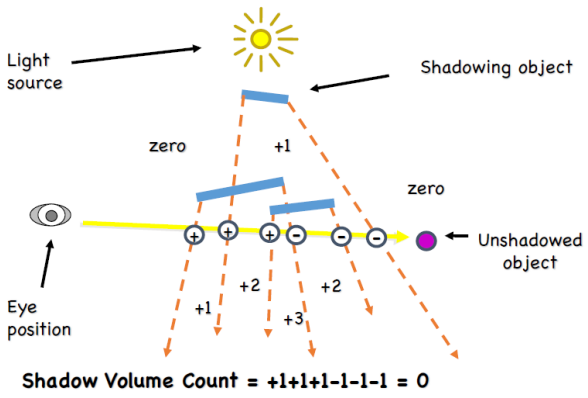


Fig. 13. Illustration of Shadow Volume Algorithm(Z-Pass).

Stencil Buffer can be added to Z-Pass Shadow Volume to make this algorithm practical for implementation, and there exist many other tricks to boost up the algorithm as well. However, it is still low in practice can be further speed up by using silhouette edges only. Silhouette edge is the edge where a back-facing polygon and a front facing polygon meet. In implementation, also shown in Figure 14,

this can be decided by computing two adjacent face normals N, N_{adj} for each polygon and doing dot product respectively with a vector that starts from a point on the edge and end at the point light's position. The green circle is the direction for computing a polygon's normal.To avoid double counting, an edge is only extruded a quad and its left face is facing towards the light.

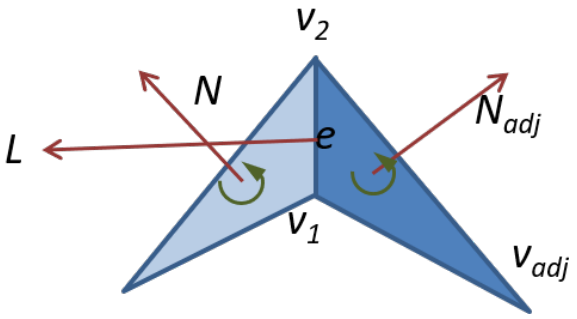


Fig. 14. Silhouette Edge Checking in Shadow Volume.

The algorithm pipeline of lighting with Z-pass Shadow Volume and Silhouette Edge is shown in Algorithm 5.

Algorithm 5 Lighting Outline with Shadow Volume

```
Render the ambient light into the color buffer
for each light source do
    Set the Stencil Buffer to 0
    Render shadow volume of silhouette edges only
    Keep the counter in the Stencil Buffer
    if Stencil Test is 0 and not in shadow then
        Blend diffuse and specular light into the final color
```

2.2.3 Comparison of Shadow Mapping and Shadow Volume. In general, Shadow Volume can create high quality shadow with an extremely high demand of computational resources, which is intolerable when generating a scene with lighting with such a complex model like Figure 15. In this case, shadow volume does much work that ends up invisible due to the complex occlusion of leaves; but the image quality still holds up well.

Shadow Mapping, on the other hand, is more efficient but creates jagged shadow edges due to under-sampling of the shadow map. A comparison of these two algorithms is shown in Figure 16, which demonstrates the analysis above.

3 EXPERIMENT

3.1 Shading: Light Attenuation

The intensity of a point light source drops off with its distance in the real world. To approximate this, we define an attenuation factor

$$f(d) = min(1, \frac{1}{0.5 + 0.01d + 0.001d^2}), \tag{8}$$

which falls off with d quadratically. Fig 17 shows the change of reflection light intensity with different distance between the object and the light source.

Fig. 15. A Failure Case of Shadow Volume w.r.t. Rendering Time.

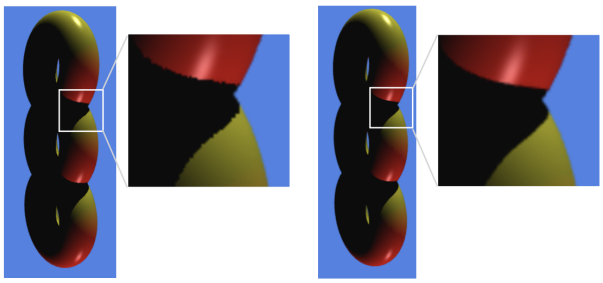


Fig. 16. Comparison of Shadow Mapping(Left) and Shadow Volume

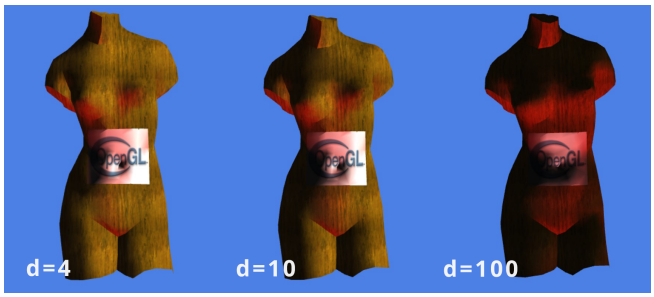


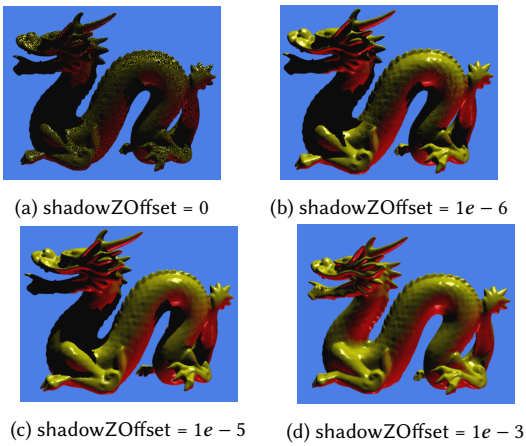
Fig. 17. Attenuation: Light intensity drops off with its distance.

3.2 Bias for Shadow Map

Bias here is the input shadowZOffset of algorithm 3: lighting with shadow mapping.

As stated above, adding shadowZOffset is a practical trick for solving floating point imprecision. As shown in (a), (b) of Figure 18, there exists discontinuity, or "Z-Fighting" when shadowZOffset is quite small, that is, shadow mapping with no or little tolerance of imprecision. When there is way too much bias, as shown in (d) of Figure 18, the shadow is little and unrealistic.

shadowZOffset is, however, hard to to tune since the best value is different for different object. Take Venus and the Dragon as an example, in Figure19, 1e-5 is suitable for the Dragon but too small



(a) shadowZOffset = 0 (b) shadowZOffset = $1e-6$

(c) shadowZOffset = $1e-5$ (d) shadowZOffset = $1e-3$

Fig. 18. Different ShadowZOffsets for Shadow Mapping.

for Venus. In practice, we can set the shadowZOffset to a relatively large number, since the "Z-fighting" effect is less tolerated to human eyes.

(a) Venus, $1e-5$ (b) The Dragon, $1e-5$

Fig. 19. Same ShadowZOffsets for Shadow Mapping in Different Objects.

4 CONCLUSIONS

Different Shading and shadowing methods have been implemented, compared and discussed in this project as shown above. For how to run the code, please refer to README for more information.