Apple Is All You Need

Evan Zhang, Taylor Wang

CS 4620 Creative Project 2

Cornell University

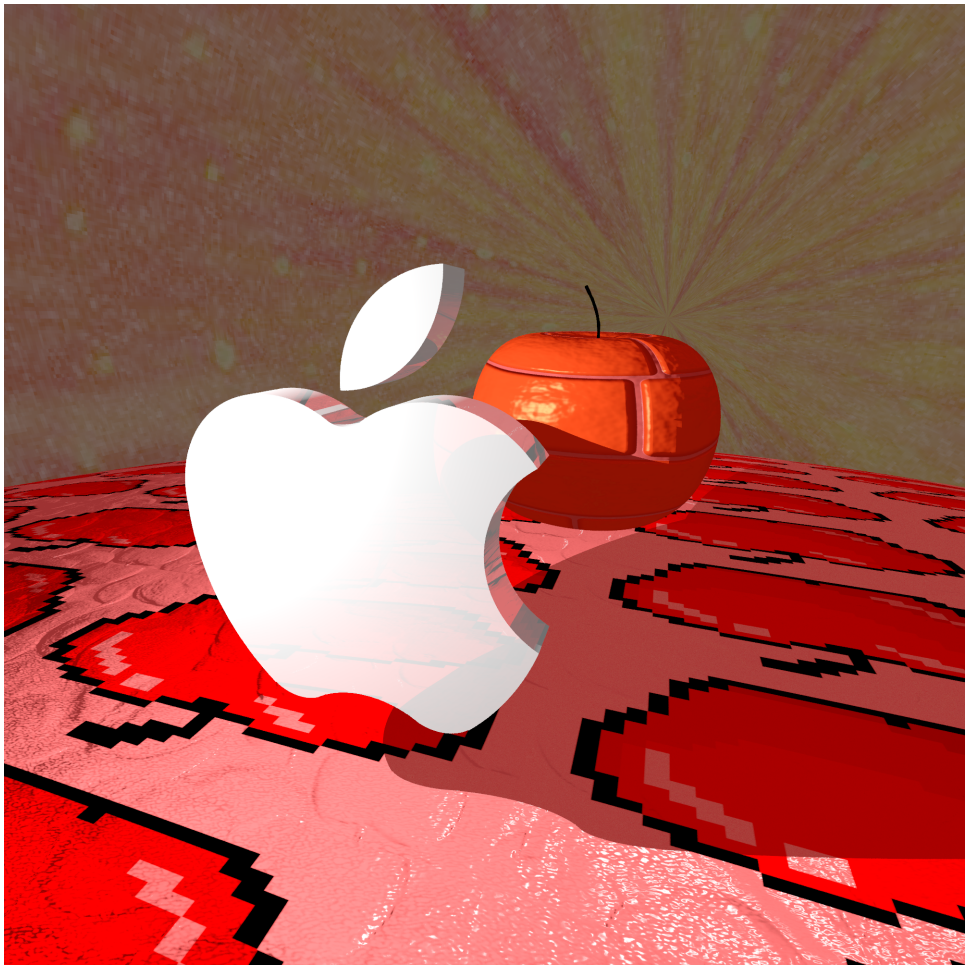

Figure 1: *Apples!*

Abstract

I was sipping apple cider when we were brainstorming for a project idea. Cider from Wegmans tastes sooo good ... Therefore, this project is all about apples. As Tim Cook always likes to emphasize, **It's something only Apple can do.** We only do apples. There are a lot of Apple elements in our rendered image: the modern looking frosted glassy Apple icon, the vintage, old-school "bricked" apple behind, the smaller apple topography and the bigger apple pixel art on the ground, and even the realistic apple texture for the background image. Almost every pixel is a part of some apple in some form.

Our final rendered image is shown in Figure 1. It is rendered at 2000 by 2000 resolution. We leveraged multiprocessing and BVH tree. It takes roughly 43 minutes to render without super-sampling and about 5 hours to render with super-sampling where the sample size is 8. Figure 1 shows the result with super-sampling.

1 Scene Composition

1.1 Apple Icon

Key features: Cylinders, Constructive Solid Geometry, Fresnel Reflection, Refraction, Bounding Volume Hierarchy

As you all probably know, the most classic Apple icon can be constructed by boolean operations upon circles of different sizes. In our ray tracer, we leverage **Constructive Solid Geometry (CSG)** tech-

niques to perform these boolean operations efficiently. However, since we are simulating 3D geometries, we apply these operations on cylinders instead of adding, intersecting, and subtracting 2D circles.

We incorporate Fresnel reflection and refraction with regular diffuse coloring to apply a frosted glassy material onto the Apple icon. **Fresnel reflection** simulates the way light reflects off surfaces at varying angles and **refraction** traces rays which pass through the icon. **Bounding Box** for the entire CSG object is constructed for faster rendering.

1.2 "Brick-ed Apple"

Key features: Normal Mapping, Texture Mapping, Blender, Mesh Loading, Bounding Volume Hierarchy

This apple is created and exported from **Blender**. We extended the OBJ file loader to extract normals and UV coordinates for each triangle, enabling detailed **Normal Mapping** and **Texture Mapping**. A normal map and its corresponding texture map of a brick wall are applied to this apple to give it a special look. **Bounding Box** for the loaded triangle mesh is constructed for faster rendering.

1.3 Floor

Key features: Normal Mapping, Texture Mapping, Bump Mapping, Bounding Volume Hierarchy

The floor of the scene is essentially a giant sphere divided into a lot of grids. We extended the Sphere class to compute the UV coordinates so that we can apply **normal, texture, and bump** maps onto each grid. The normal map, generated using an online tool, simulates surface normals to create the topography of small "apple hills". The texture map features the pixel art of an apple, providing a base color and pattern that enhances visual interest. The bump map introduces subtle surface irregularities, adding tactile realism without increasing geometric complexity. Certainly, **bounding Box** for the sphere is constructed for faster rendering.

1.4 Background Image

Key features: Background Image Mapping

Yes. Even the background image is related to apples. We use a texture map for a real apple and map it from Cartesian coordinates to spherical coordinates. This is simulating folding the texture map around the upper edge. This creates a cool "time tunnel" effect since there are a lot of yellow vertical stripes on the texture map.

2 Features

2.1 Constructive Solid Geometry

CSG is primarily used to create the Apple icon. We draw the circles on top of the Apple icon and measure their relative sizes. We define a Cylinder class and a RectangularBox class to construct the icon. The cylinder is the most important added primitive in this project. A cylinder is defined by its base center, axis direction, height, and radius. Comparatively, RectangularBox is defined purely for auxiliary purposes, which will be introduced later. In total, there are 11 cylinders and 3 auxiliary rectangular boxes. All have the same height.

The "ideal" process to construct a 2D Apple icon is shown in graphs 1-6 in Figure 2. The reason why I call this process ideal is because it requires more complicated operations other than simple union, difference, and intersection. It needs some kinds of disintegration and deletion, which our implementation does not have. More specifically, the red area A in graph 2 and the red area B in graph 3 can not be constructed with union, difference, and intersection. This is why we need rectangular boxes to carve these areas out.

Solve for A: Graph 7 shows an alternative way to do the process in Graph 2. Two cylinders (green), one small rectangular box (red), and one big rectangular box (purple) are used to achieve this. The upper curvature of Area A is basically the small rectangular box **minus** a cylinder.

Solve for B: Graph 8 shows an alternative way to do the process in Graph 3. One additional rectangular box is needed to achieve this. The two lower curvatures of Area B are simply the rectangular box **minus** two cylinders.

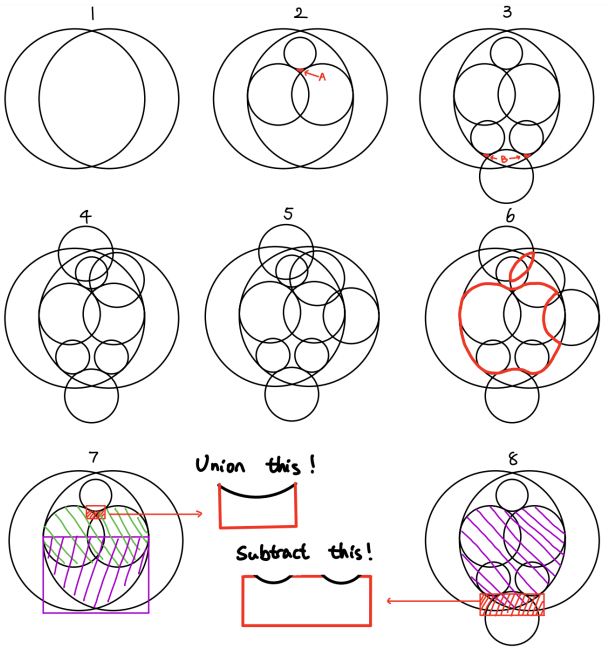

Figure 2: *Boolean operations between cylinders and rectangular boxes to shape the Apple icon*

2.2 Fresnel Reflection

To accurately simulate the behavior of light at material interfaces (especially the glassy Apple icon), we implemented Fresnel Reflection using Schlick's approximation. In our implementation, when a ray intersects a surface, we first determine whether it is entering or exiting the material to correctly compute the ratio of refractive indices (η). Using the angle between the incoming ray and the surface normal (θ_i), we apply Schlick's formula to estimate the reflectance ($\boldsymbol{R}$), which dictates the intensity of the reflected light. We also accounted for total internal reflection when the angle of incidence exceeds the critical angle even though it is extremely rare in our scene.

In class, we discussed how reflection dominates over refraction when the angle of incidence is nearly 90 degrees. We observe this phenomenon in our scene as well. As shown in Figure 3, on the lateral (side) area of the Apple icon where the angle of incidence is big, we see a lot of reflection of the "brick-ed" apple.

Figure 3: *Reflection*

2.3 Refraction

Since we have already calculated reflectance ($\boldsymbol{R}$), transmittance is just 1-$\boldsymbol{R}$. Also, utilizing the angle of incidence (θ_i) between the incoming ray and the surface normal, we apply Snell's Law to compute the angle of refraction and subsequently derive the direction of the refracted ray.

Opposite from reflection, when the angle of incidence is small, we should see more refraction. As shown in Figure 4, on the cap (front) of the Apple icon where the angle of incidence is small, we see a lot of refraction of the floor.

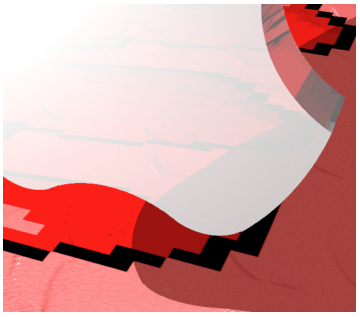

Figure 4: *Refraction*

2.4 Normal Mapping

Normal maps are applied on both the "brick-ed" apple and the floor. During rendering, the UV coordinates computed for each intersection point are used to bilinearly sample the corresponding normal from the normal map stored in the Material class. Figure 5 shows the normal map for the floor, and Figure 6 shows the normal map for the "brick-ed" apple. This sampled normal, initially defined in tangent space, is then transformed into world space using the surface's tangent and bitangent vectors. By adjusting the original geometric normal with these perturbed normals, the lighting calculations produce the illusion of detailed surface features.

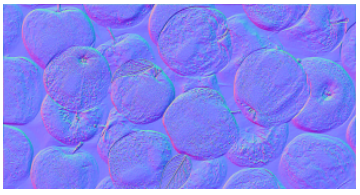

Figure 5: *Normal map for the floor*

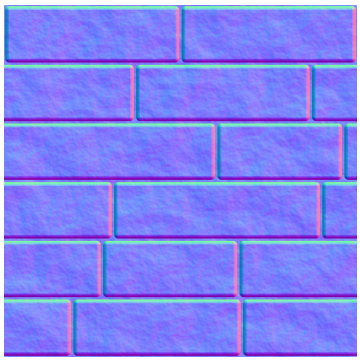

Figure 6: *Normal map for the "brick-ed" apple*

Figure 7 shows the "apple hills" topography after applying normal map to the floor sphere. Figure 8 shows the "brick-ed" effect of the apple.

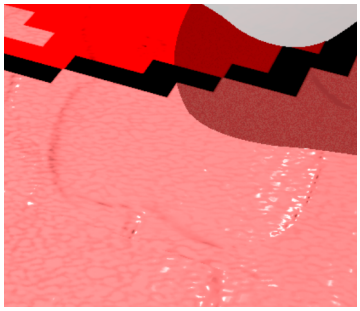

Figure 7: *Apple topograph on the floor*

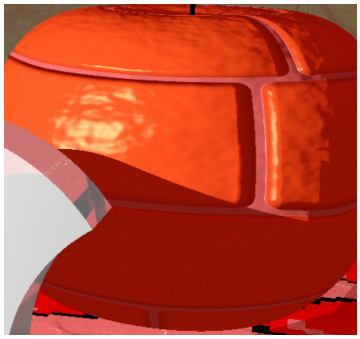

Figure 8: *"Brick-ed" effect*

2.5 Texture Mapping

To enrich the visual detail of the floor and the "brick-ed" apple, we implemented texture mapping by mapping 2D image textures onto 3D surfaces. This process begins with computing UV coordinates for each intersection point. For instance, in the Sphere class, we convert Cartesian coordinates to spherical coordinates to derive the UV mappings. The Material class is equipped with a diffuse map, which holds the texture image data. During rendering, when a ray intersects a surface, the corresponding UV coordinates are used to perform bilinear sampling of the diffuse texture, retrieving the precise color information from the texture image. This sampled color is then utilized as the diffuse color in the shading calculations. Fig-

ure 9 shows the texture map for the floor, and Figure 10 shows the texture map for the "brick-ed" apple.

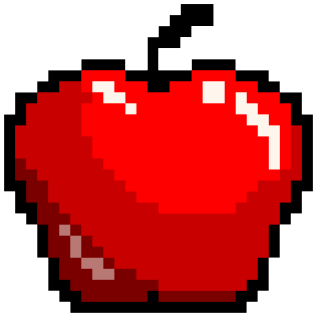

Figure 9: *Texture map for the floor*

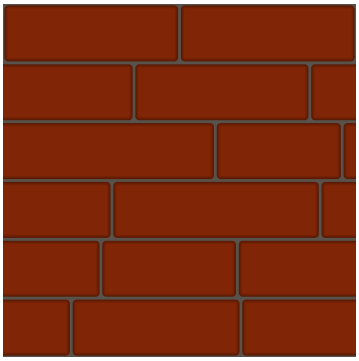

Figure 10: *Texture map for the "brick-ed" apple*

2.6 Bump Mapping

Bump mapping utilizes a grayscale texture (bump map) to simulate small-scale surface irregularities by perturbing the surface normals during shading calculations. In our implementation, only the floor sphere has a bump map image stored in the Material class. We implemented bump mapping for other primitives but decided not to use them. The bump map is bilinearly sampled at the intersection point using the sphere's UV coordinates. The sampled bump value is then scaled by a predefined bump scale factor to determine the height displacement. Utilizing the surface's tangent and bitangent vectors, we adjust the original geometric normal by adding the scaled perturbations. Figure 11 shows the bump map for the floor.

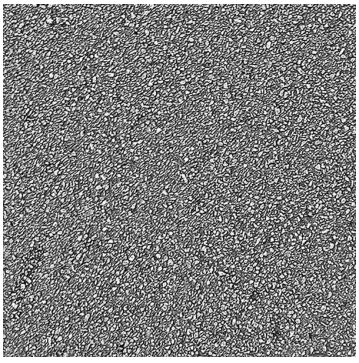

Figure 11: *Bump map*

Figure 12 shows the floor after the bump map is applied.

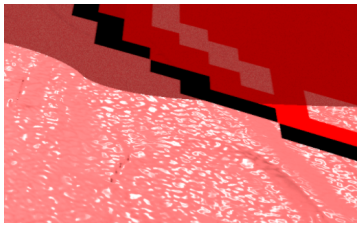

Figure 12: *Bump map*

2.7 Bounding Volume Hierarchy

We implemented axis-aligned bounding boxes (AABB) for all primitives in our scene. Then we constructed a BVH tree to improve render efficiency. Each node in the BVH tree encapsulates a subset of objects within the bounding box. By recursively splitting the scene along the axis with the largest extent, the BVH efficiently partitions space, allowing rays to quickly eliminate large regions without intersections. During rendering, rays traverse the BVH. They first check against the top-level bounding boxes and only perform detailed intersection tests with objects in intersected nodes.

Figure 13 shows an example scene that we created to test BVH's performance improvement. The scene is composed of 10x10x10 = 1000 spheres and a large floor. Rendering with BVH takes **148.31** seconds whereas rendering without BVH takes **2245.30** seconds. That is, in a scene with a lot of occlusions between objects, our implementation of the BVH tree demonstrates a **15x** performance improvement.

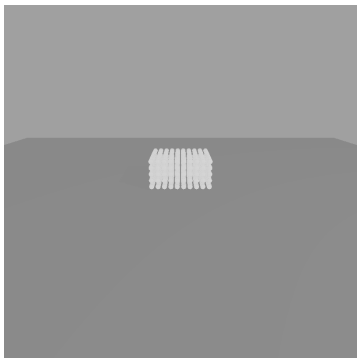

Figure 13: *BVH efficiency test scene*

2.8 Defocus Blur

Our implementation utilizes the camera's aperture and focal distance parameters to control the extent of the blur. When a ray is generated, if defocus blur is enabled, its origin is randomly sampled within the camera's aperture area. The direction of the ray is then adjusted to converge towards a predetermined focal point. This ensures that objects at the focal distance remain sharp while those closer or further away become progressively blurred. This approach not only adds depth to the scene but also highlights the main subjects by subtly de-emphasizing background and foreground elements. Additionally, super-sampling is employed to enhance image quality by averaging multiple samples per pixel, reducing noise and aliasing artifacts. As shown in Figure 14, the focal point lies on the right side of the Apple icon. Therefore, the left side of the icon, the "brick-ed" apple behind and the floor are blurry. This image is also rendered at 2000 by 2000 resolution. Bust since super-sampling is

incorporated, the time it takes to render this image grows proportionally to the number of rays we sample per pixel.

Figure 14: *Defocus blur for our scene*

2.9 Super-sampling

Since the floor has a grid-like pattern of the apple pixel art, the original rendered image suffers from the problem of aliasing on the far end of the floor, as shown in Figure 15. Therefore, we incorporated super-sampling to perform anti-aliasing. The result is shown in Figure 16.

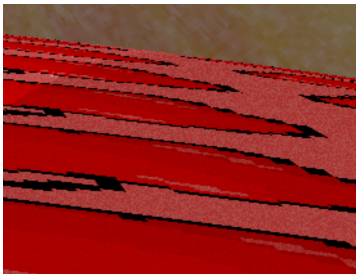

Figure 15: *Alias artifact*

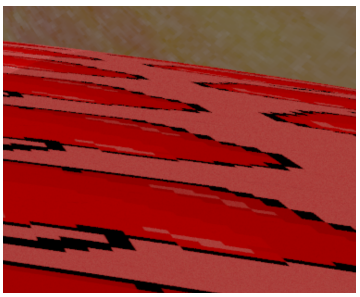

Figure 16: *Anti-alias*

2.10 Background Image Mapping

The background is sampled from the texture of a real apple as shown in Figure 17. We created a "time tunnel" effect with this texture image by simulating wrapping around along the top image boundary. This mapping is achieved by converting the ray's Cartesian direction vector into spherical coordinates, which are then normalized to obtain UV texture coordinates. Additionally, a darkening factor is applied to the bilinearly sampled background color. This allows for subtle control over the background's brightness and its influence on the overall scene ambiance.

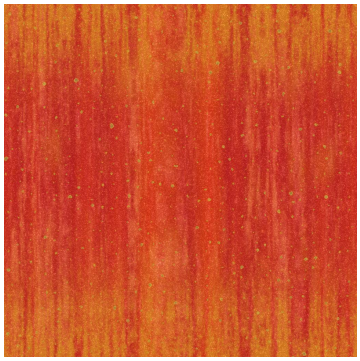

Figure 17: *Background image before projection*

Figure 18 shows the "time tunnel" background in the rendered image.

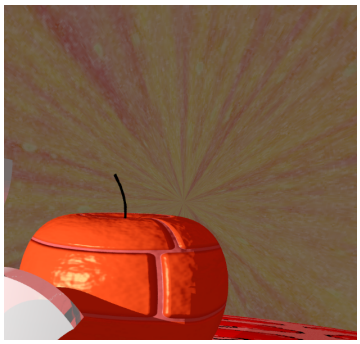

Figure 18: *Background image after projection*

3 References

- Website to create the normal map: https://cpetry.github.io/NormalMap-Online/
- Website to download texture and normal maps: https://opengameart.org/
- Project name: Vaswani, Ashish, et al. *Attention Is All You Need*. Advances in Neural Information Processing Systems, vol. 30, pp. 5998–6008, 2017.