Gabriel Fuchs (gdf42)
CS 4620
Quimey Moure
Professor Davis
C2 Writeup

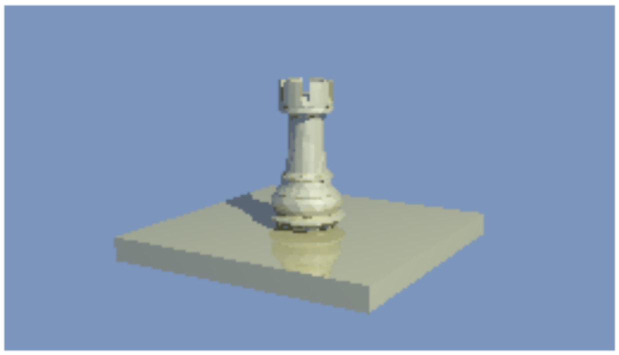
Preface

The following writeup includes a procedural overview of the additional features added to the ray tracer (starting from the result of A4) throughout the C2 creative process. The images associated with each section do not necessarily relate at all to the output image and instead simply showcase the additional feature added.

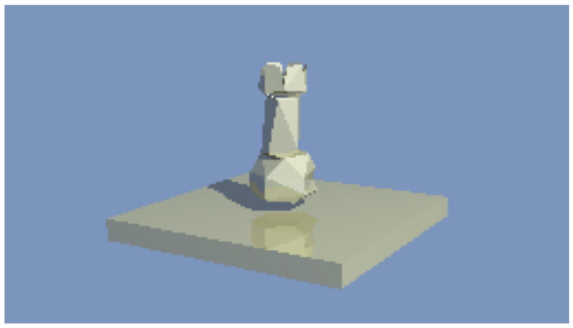
Importing objs from Blender

By default the previous loader was able to load all triangles into the scene but lacked the ability to distinguish between objects in order to color various objects differently. The standard load function is seen below with a chess piece [imported model] of different triangle counts.

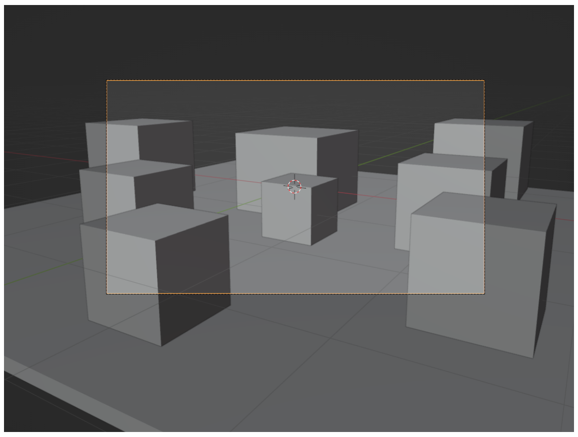
```
CPU times: total: 7min 31s
Wall time: 7min 31s
```



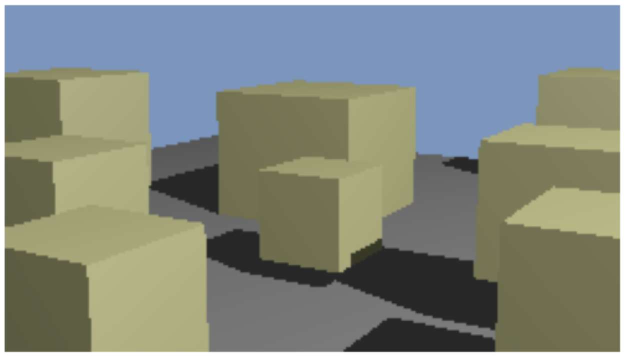
```
CPU times: total: 50 s
Wall time: 50 s
```



Below is the result of the addition of the function *Read_obj_extended* in utils. The left image is the blender scene that was imported (with matching camera position and orientation) and on the right is the ray tracer generated image with different object colors to show object separation.



```
CPU times: total: 1min 9s
Wall time: 1min 9s
```

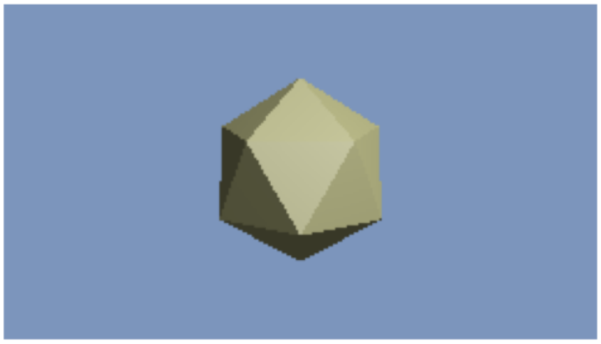


Smoothing normals between triangles

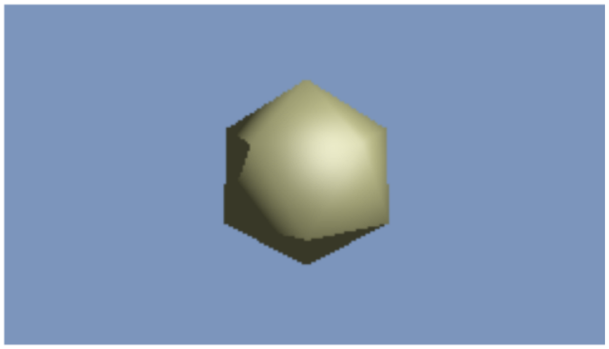
When exporting object files from blender, each vector has an associated outward facing normal. We can use a form of linear interpolation in terms of the barycentric coordinates (already calculated for collision direction to create) in order to smooth the normals for light reflection across a surface of triangles. The *Read_obj_extended* function described above sorts the normals based on vertex to be utilized within this context.

IcoSphere before and after smoothing:

```
CPU times: total: 8.19 s
Wall time: 8.2 s
```

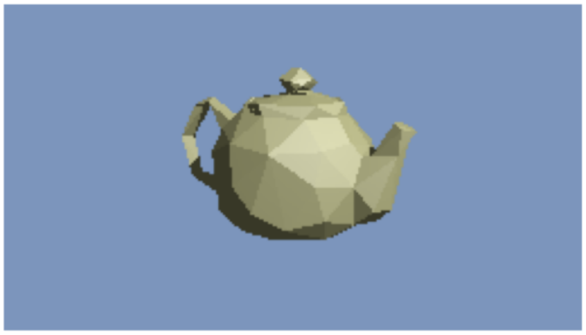


```
CPU times: total: 8.27 s
Wall time: 8.26 s
```

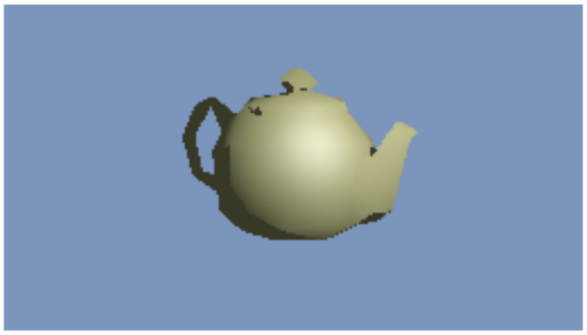


Teapot before and after smoothing:

```
CPU times: total: 1min 18s
Wall time: 1min 18s
```

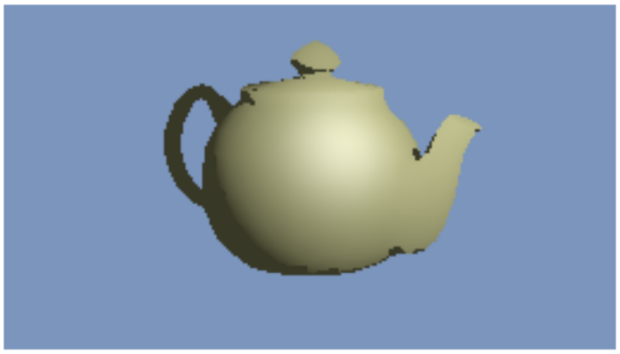


```
CPU times: total: 1min 19s
Wall time: 1min 19s
```



Smoothing applied to high poly count teapot:

```
CPU times: total: 8min 13s
Wall time: 8min 13s
```



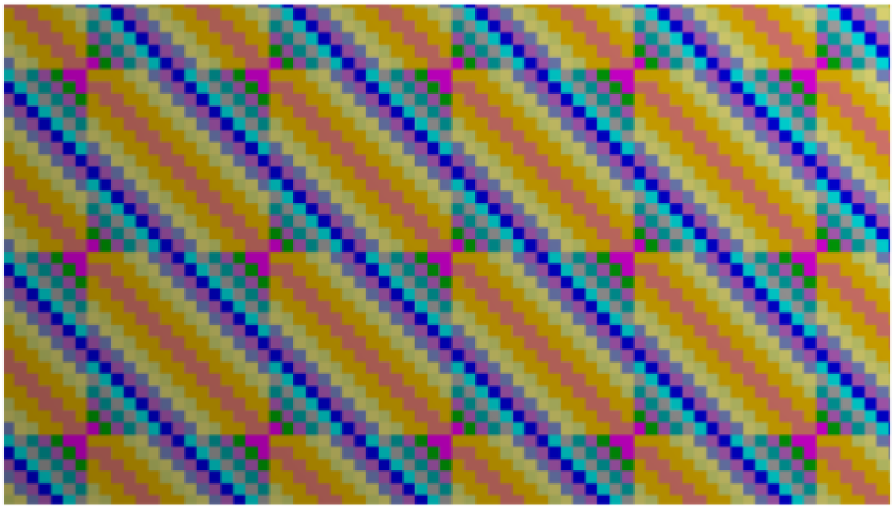
Note that artifacts are developed on object edges due to a combination of shading issues during abrupt normal change as well as due to overall low resolution in ray tracing. We also lose detail in areas that should not be smoothed. It is possible but tedious to choose what parts [which triangles] of an object should be smoothed.

Texture maps

Texture maps were implemented using the uv mapping accounted for in blender obj file imports. A new class called texture was created which handles this feature and can be passed into a material object. Currently, only triangles generated from obj files are supported by textures due to a lack of uv generating functions on the primitives.

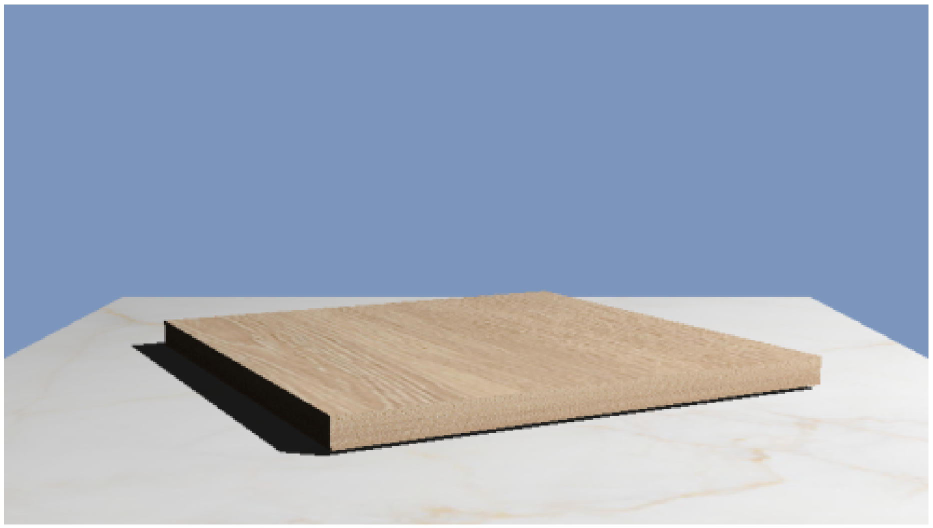
Simple Pattern scaled down to show repetitions

```
triangles to calculate: 2
CPU times: total: 39.1 s
Wall time: 39.1 s
```



A wood cutting board and marble table: the start to the final image submission

```
triangles to calculate: 14
CPU times: total: 1min 44s
Wall time: 1min 44s
```

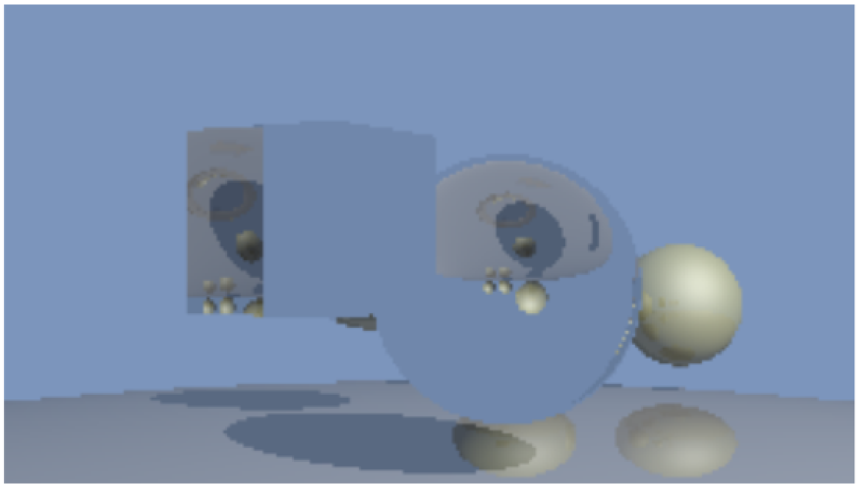


Refraction

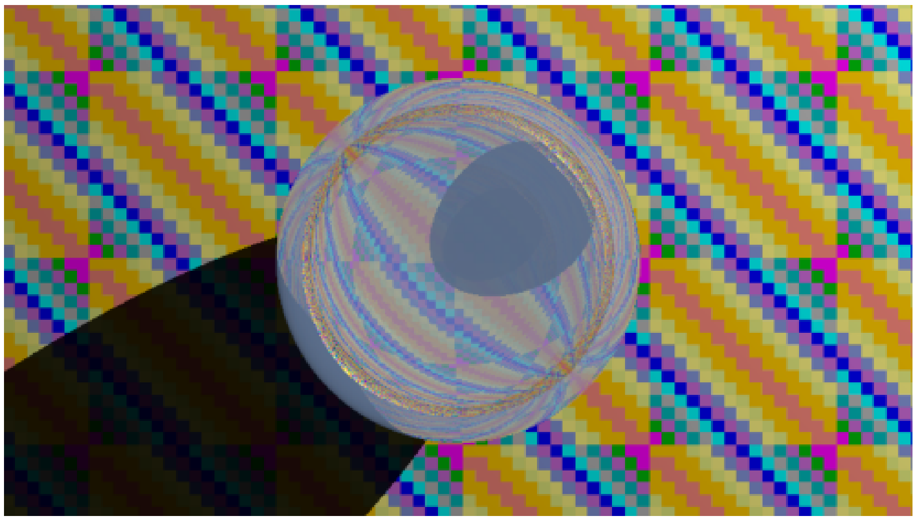
Refraction was implemented physically through following the rules of Snell's law. Total internal reflection was accounted for.. The refraction function, similar to the standard mirror shading function, casts a limited amount of rays (recursively). This computation becomes rather expensive fast and as a result the recursion depth was limited.

Glass Cube and Ball: Refraction without mirroring:

```
CPU times: total: 56.2 s
Wall time: 56.3 s
```



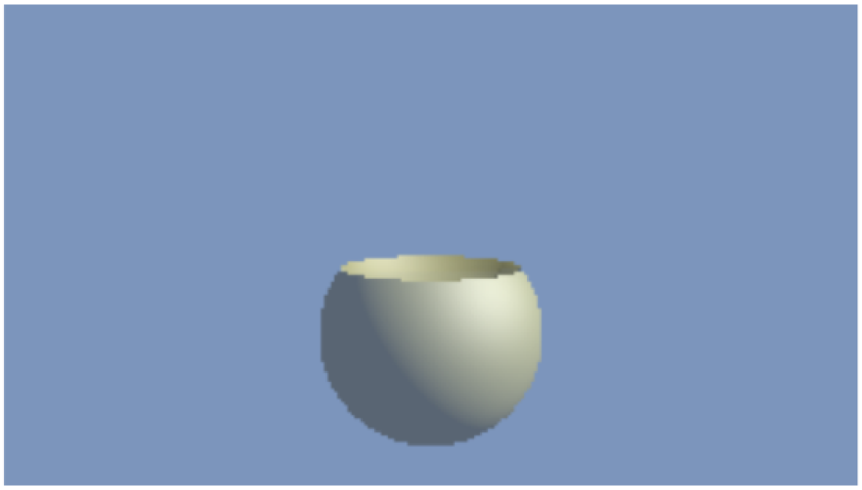
Top view of glass Ball: Refraction with an implementation of fresnel reflection.



Constructive Solid Geometry

CSG supports union, intersection, and difference and can have as many binary levels as desired. One challenge that had to be overcome within this implementation was making sure shadows worked properly; particularly when using the difference operator. Additionally, all intersect functions take a parameter *csg* in order to decide whether to return just the first hit or all hits (all hits are needed for CSG computations).

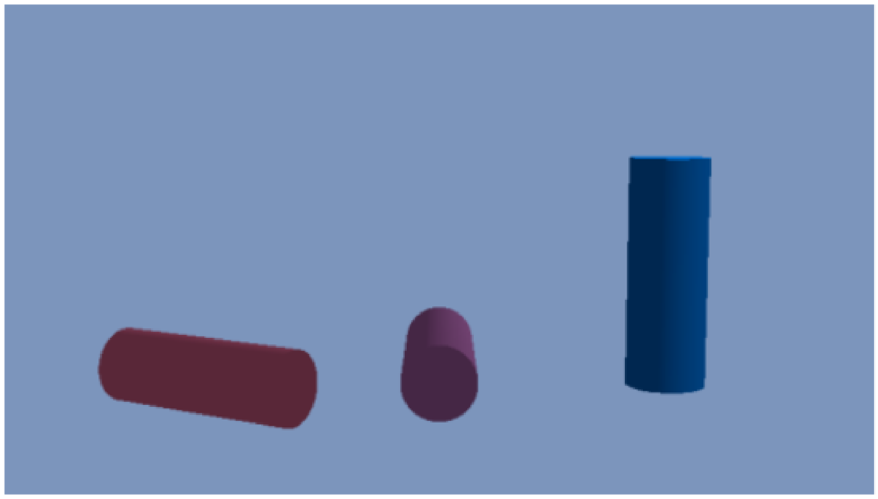
```
CPU times: total: 1.56 s
Wall time: 1.54 s
```



Cylinder Primitives

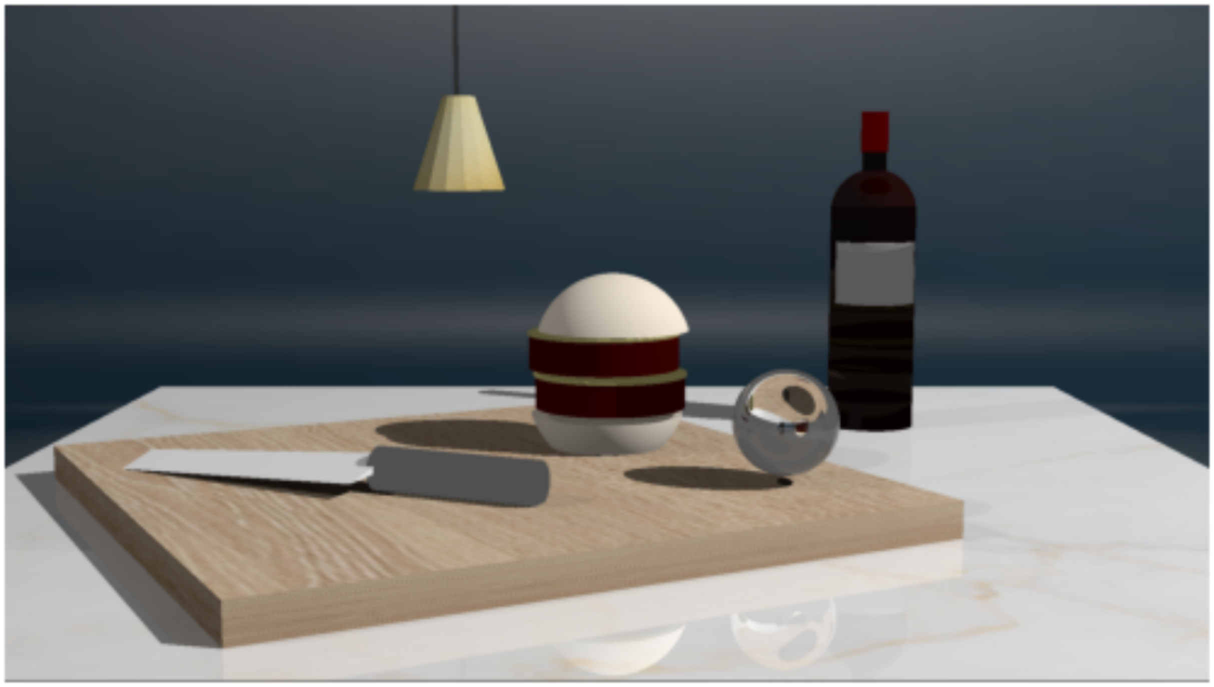
Cylinder primates were added. Position changes and rotation of the cylinders was handled by manipulating the direction and origin of incoming rays rather than the intersection of a manipulated cylinder directly.

```
CPU times: total: 1min 20s
Wall time: 1min 20s
```



Submission:

```
triangles to calculate: 45
CPU times: total: 2h 28min 54s
Wall time: 2h 29min 23s
```



The scene depicts a dinner table. There is a cutting board with a chef's knife, hamburger, and wine bottle. Additionally, there is a light hanging in the background and a glass marble is included to depict the implementation of refraction.

- **Hamburger:** CSG using cylinders and spheres (operations include difference and union)
- **Cutting Board:** Triangles imported from blender and textured
- **Table:** Plane with a texture
- **Wine bottle:** CSG using 4 cylinders and a sphere (all operations were union so CSG was not required here)
- **Hanging Light**: Triangles imported from blender and textured
- **Glass Marble**: Refraction
- **Background**: Plane with a texture

The final render was with the resolution of 1792 by 1008. Given the 45 triangles and 15+ other primitives with both refraction and mirroring in play, the total rendering time was nearly 2 and a half hours.