Helios

Realtime ray tracing in CUDA

Aditya Singh Rathore,
2018007
GPU Computing
CSE560, Winter, 2022

Milestones

- Ray-object Intersection

Logistic Setup

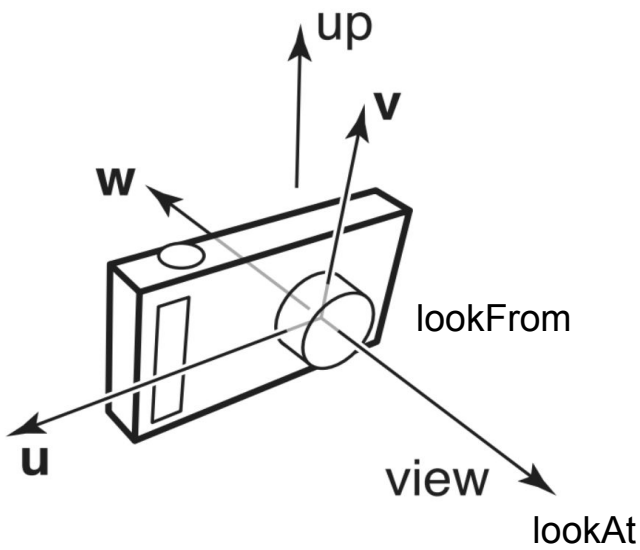
- OpenGL Setup
- Cuda-OpenGL data sharing
- Mesh setup

Overview of Algorithm

- Create a full screen quad in OpenGL
- Create a texture.
- Draw over the texture from cuda kernel
- Draw the texture over the Quad
- For real time,
  - Create a camera.
  - Control position and target of camera
  - Pass camera parameters to Kernel and keep updating

Camera

- We can control lookFrom and lookAt using Keyboard and mouse
- Ray Origin
- Ray Direction



Texture

- Create a OpenGL **GL_TEXTURE_2D** m_textureID
- Create a cudaGraphicsResource * and register the OpenGL texture with cudaGraphicsGLRegisterImage
- Map the texture to a cudaArray using cudaGraphicsMapResources
- Create a cudaResourceDesc (CUDA Resource descriptor) with the array.
- Create a cudaCreateSurfaceObject from resource descriptor.

Data Representation

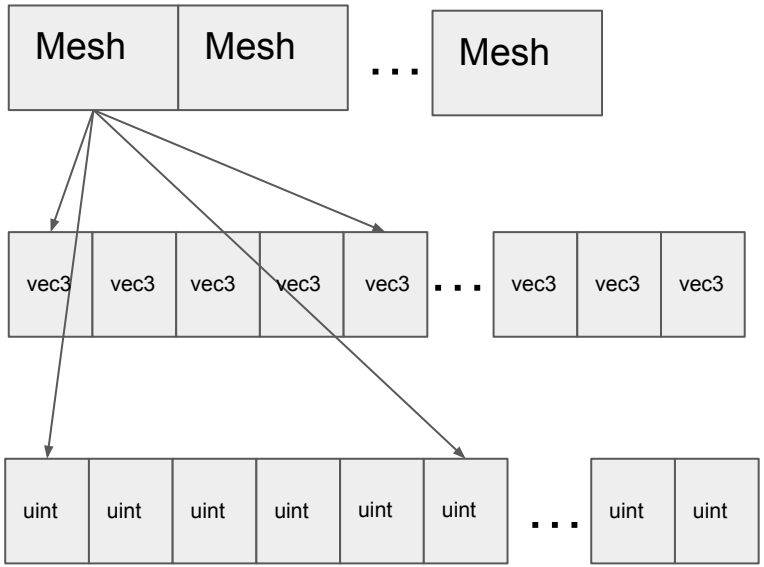
- Used assimp to read obj files.
- Model
  - Mesh
    - Vertices
    - Indices
  - Mesh
    - …
  - …

- Pointer of Pointer of Pointers.

```
class Vertex{
  glm::vec3 Position;
}

class Mesh{
  vector<Vectex> vertices;
  vector<unsigned int> indices;
}

class Model{
  vector<Mesh> meshes;
}
```

Data Representation

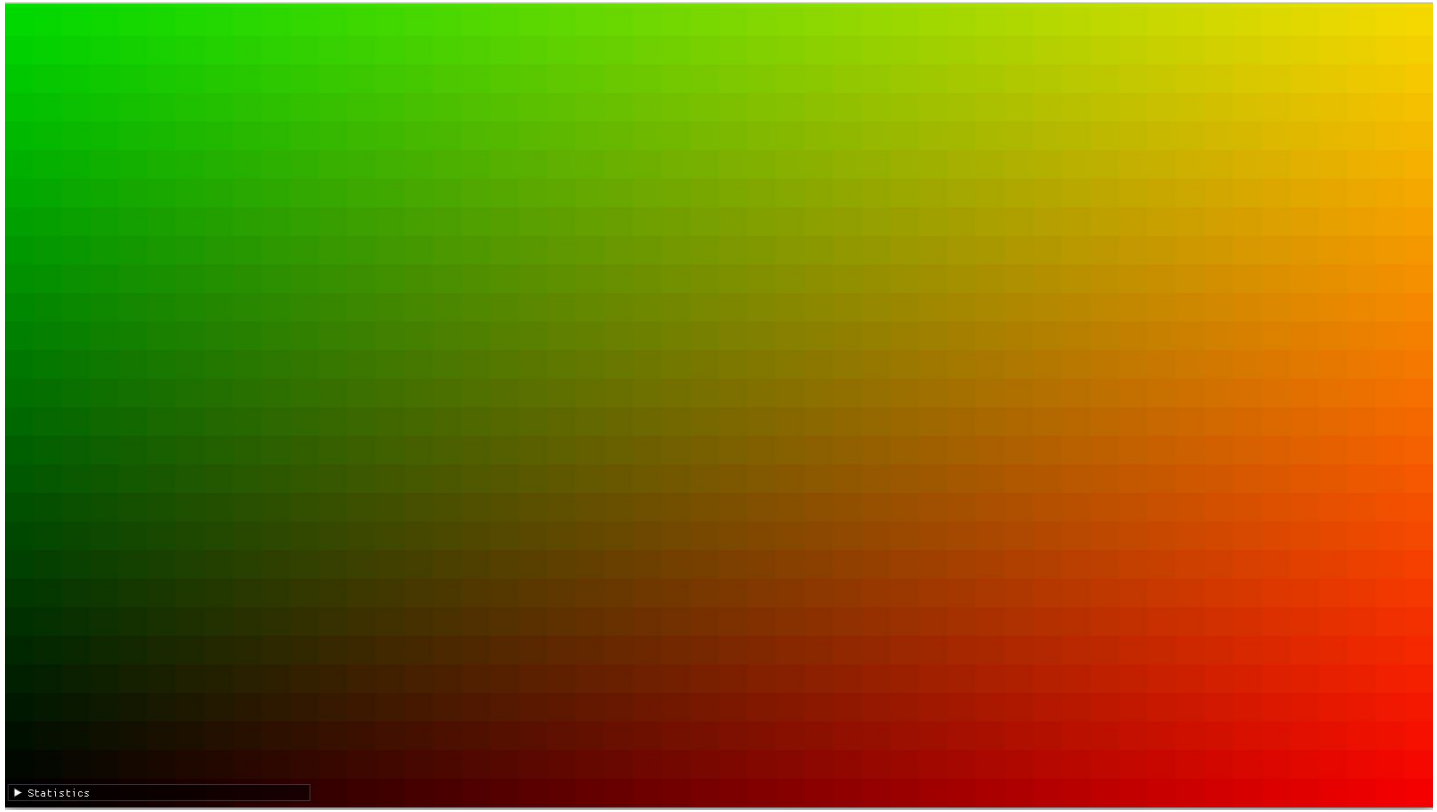
- Flattened Object into Arrays




```
struct Mesh{
    int start_vertices;
    int num_vertices;
    int start_indices;
    int num_indices;
};

struct Mesh* meshes;
glm::vec3* vertices;
unsigned int* indices;
```

Kernel

- A thread for each pixel

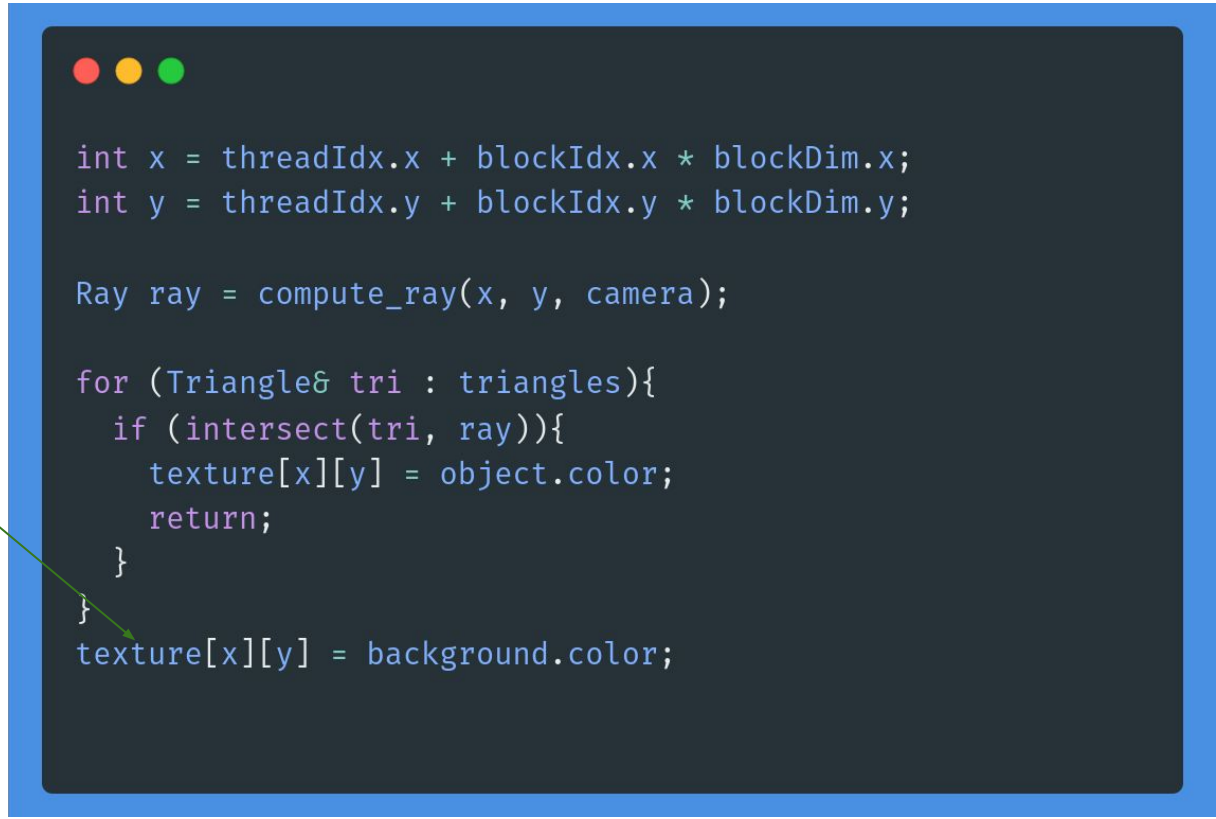


Kernel

```
int x = threadIdx.x + blockIdx.x * blockDim.x;
int y = threadIdx.y + blockIdx.y * blockDim.y;

Ray ray = compute_ray(x, y, camera);

for (Triangle& tri : triangles){
  if (intersect(tri, ray)){
    texture[x][y] = object.color;
    return;
  }
}
texture[x][y] = background.color;
```

Kernel

`cudaSurfaceObject_t`

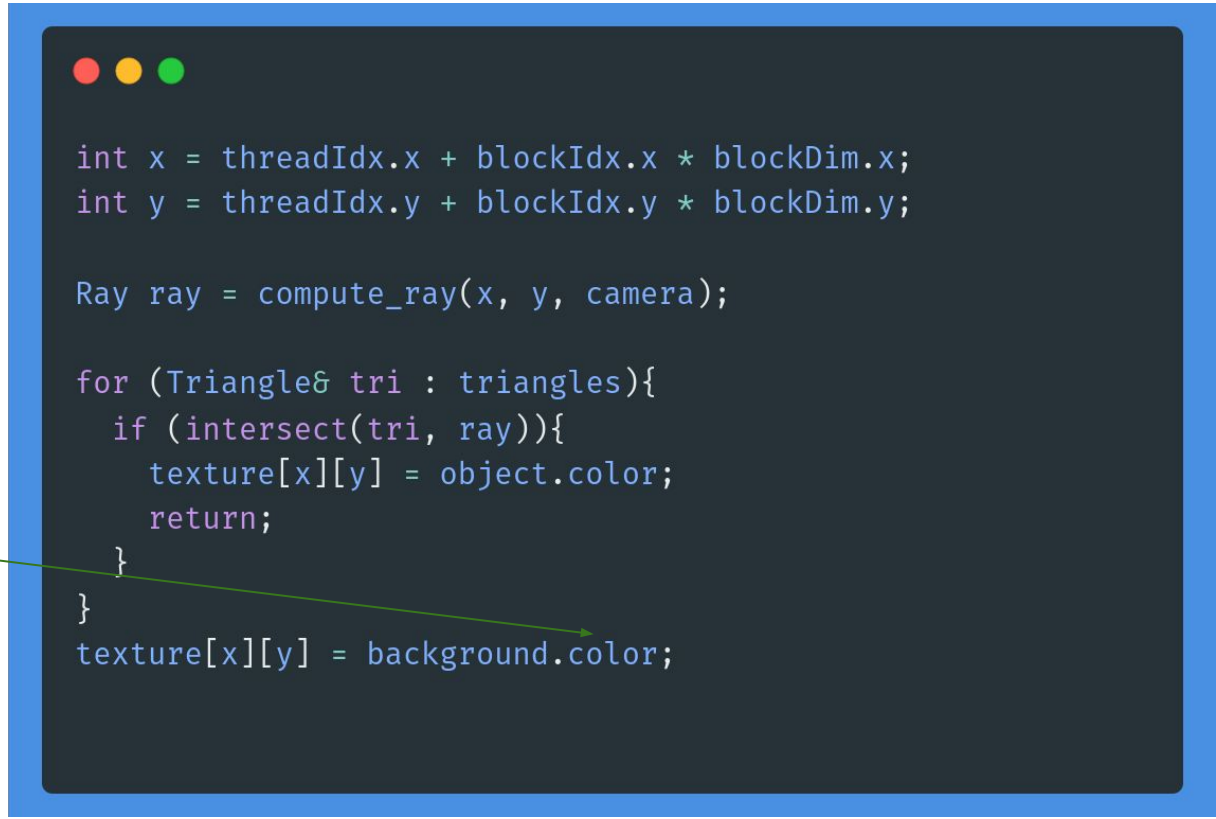


```
int x = threadIdx.x + blockIdx.x * blockDim.x;
int y = threadIdx.y + blockIdx.y * blockDim.y;

Ray ray = compute_ray(x, y, camera);

for (Triangle& tri : triangles){
  if (intersect(tri, ray)){
    texture[x][y] = object.color;
    return;
  }
}
texture[x][y] = background.color;
```

Kernel

cudaSurfaceObject_t

uchar4
(r,g,b,a)



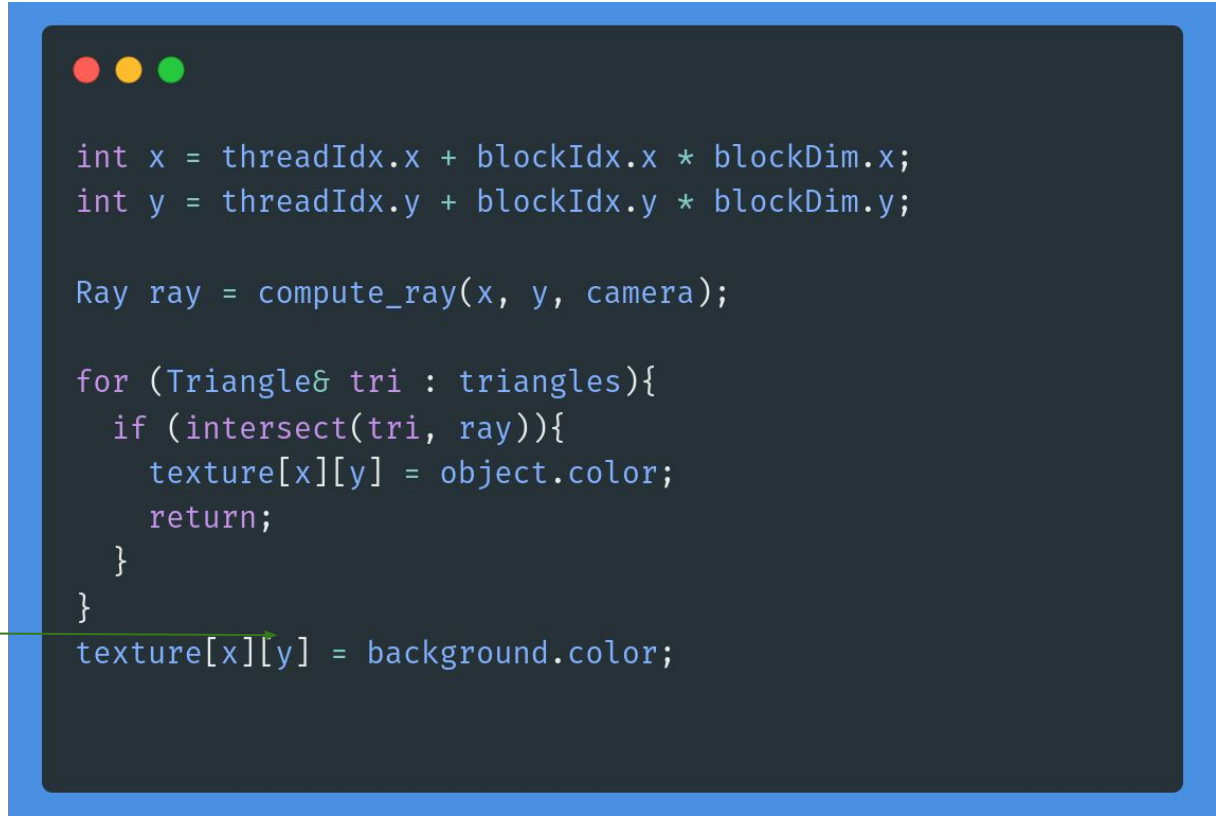
```
int x = threadIdx.x + blockIdx.x * blockDim.x;
int y = threadIdx.y + blockIdx.y * blockDim.y;

Ray ray = compute_ray(x, y, camera);

for (Triangle& tri : triangles){
  if (intersect(tri, ray)){
    texture[x][y] = object.color;
    return;
  }
}
texture[x][y] = background.color;
```

Kernel

cudaSurfaceObject_t

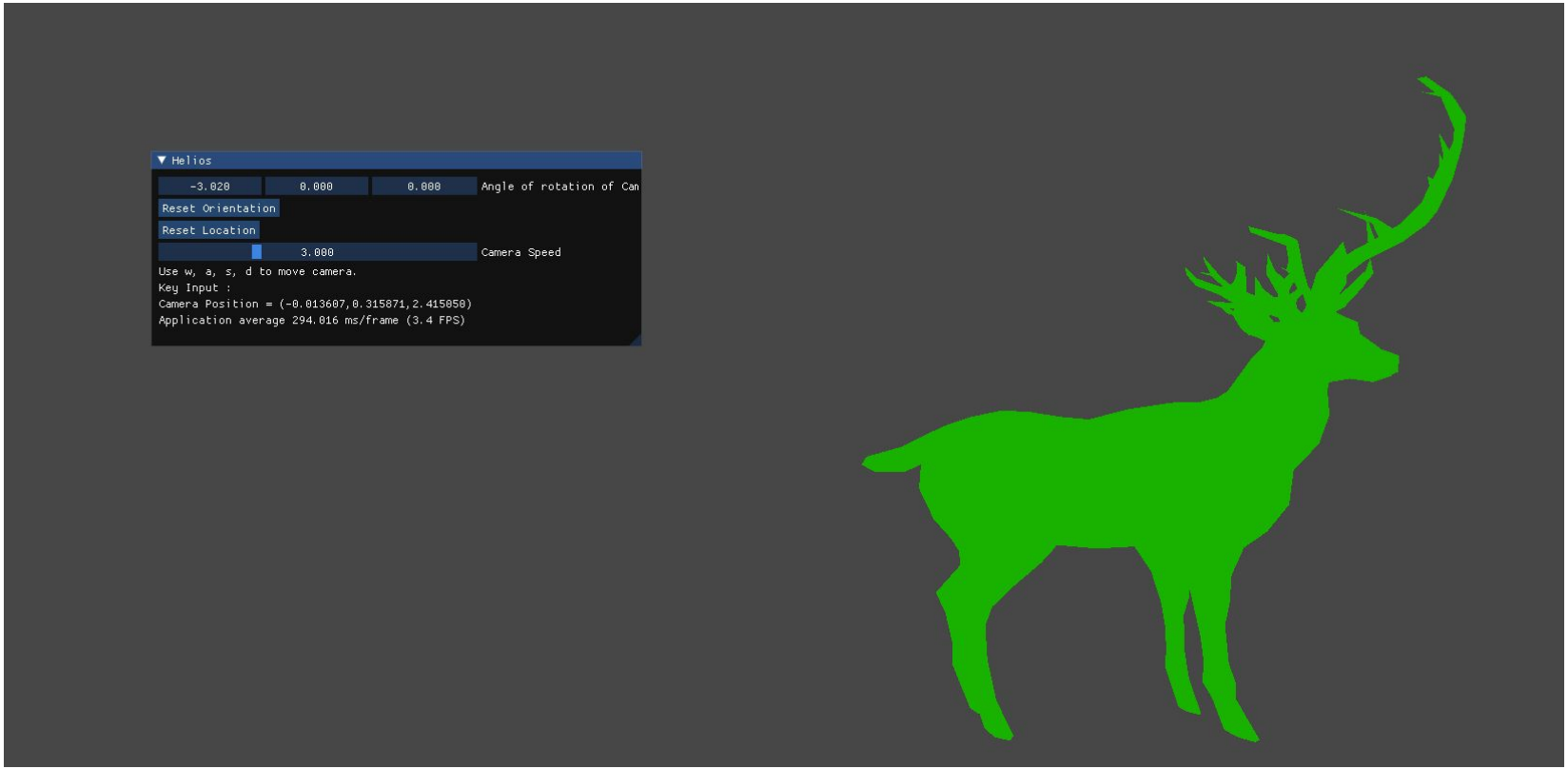
uchar4
(r,g,b,a)

surf2Dwrite



```
int x = threadIdx.x + blockIdx.x * blockDim.x;
int y = threadIdx.y + blockIdx.y * blockDim.y;

Ray ray = compute_ray(x, y, camera);

for (Triangle& tri : triangles){
  if (intersect(tri, ray)){
    texture[x][y] = object.color;
    return;
  }
}
texture[x][y] = background.color;
```

Results


Helios
-3.020
0.000
0.000
Angle of rotation of Can
Reset Orientation
Reset Location
3.000
Camera Speed
Use w, a, s, d to move camera.
Key Input :
Camera Position = (-0.013607,0.315871,2.415050)
Application average 294.016 ms/frame (3.4 FPS)

Results

Model	Vertices	Faces	Avg. Frame Rate	Meshes
Cube	24	12	280	1
Low Poly Tree	540	280	20	2
Deer	4186	1508	4.7	1
Sofa	1588	1732	4	12
Sword	2313	2596	3	4
Low Poly Car	7945	5172	Couldn’t Run	31
House	14064	16897	Couldn’t Run	41
Backpack	53464	67907	Couldn’t Run	79

Bottlenecks

- Iterating over all objects.
  - As the number of vertices increases, frame rate drops.
  - As we go away from object, frame rate drops.

- Ray-Triangle Intersection.
  - Currently using Geometric Method
  - Faster methods exist

Bottlenecks (nvprof)

Type	Time(%)	Time	Calls	Avg	Min	Max	Name
GPU activities	100.00%	21.3397s	98	217.75ms	1.1652ms	292.85ms	ray_trace
API calls	99.36%	21.3648s	98	218.01ms	1.3611ms	292.90ms	cudaDeviceSynchronize
API calls	0.62%	134.26ms	1	134.26ms	134.26ms	134.26ms	cudaGraphicsGLRegisterImage
API calls	0.02%	3.2407ms	98	33.068us	13.403us	145.07us	cudaLaunchKernel
…	…	…	…	…	…	…	…

Future Work

- Better way to iterate over objects
  - BVH algorithm
- Faster Ray-Triangle intersection
  - Möller-Trumbore algorithm
- Shading and lighting
  - Blinn-Phong Shading