

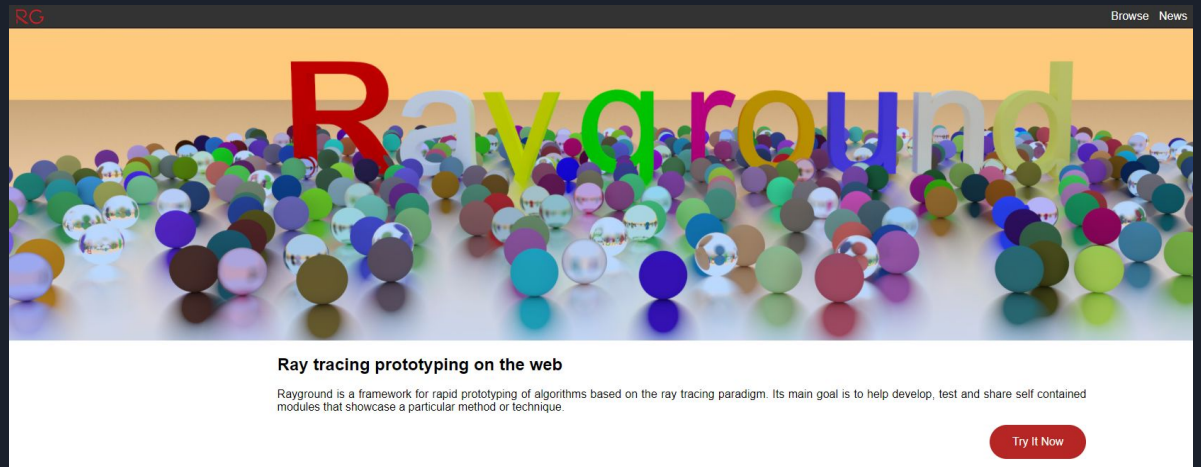
# Rayground:

An Online Educational Tool for  
Ray Tracing

TDT03 - Presentation by Martin Solheim.

# What is Rayground

- An educational tool to learn Ray Tracing
  - Made by: N. Vitsas , A. Gkaravelis , A. A. Vasilakis , K. Vardis , G. Papaioannou
  - In: Athens University of Economics and Business
- An online IDE for ray tracing algorithms





# Why

- To learn ray tracing without distractions
  - The specific characteristics of a framework/ API, etc
- Learning ray tracing in OpenGL/c++ has a lot of additional challenges
- Low-level APIs like OptiX can be quite technical
- Argues that learning about ray tracing should be more accessible
  - Webapps are super accessible

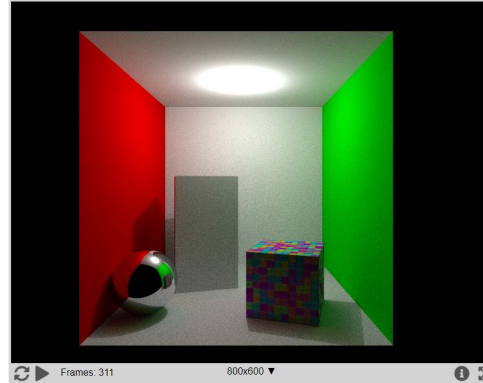
# How

- Online IDE for ray tracing algorithms
- Can run in any WebGL2 compliant browser
- Inspired by ShaderToy
- User is encouraged to use GLSL functions

and types

- Vec, mat
- cross(), dot()

Cornell with procedural texturing with mouse

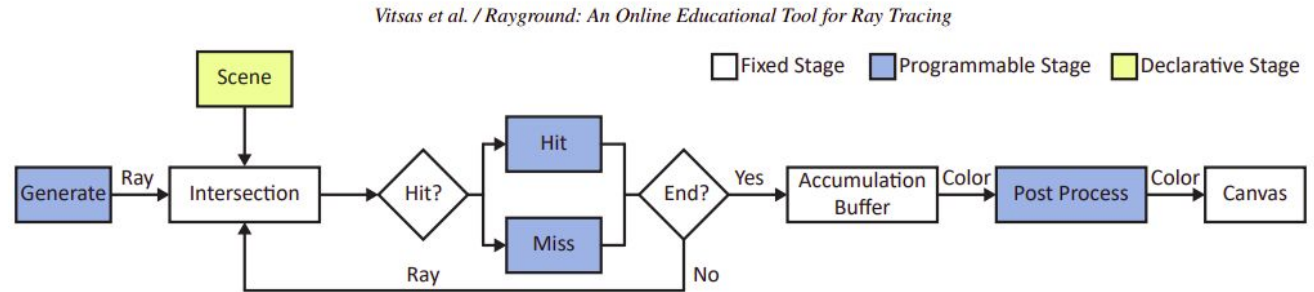
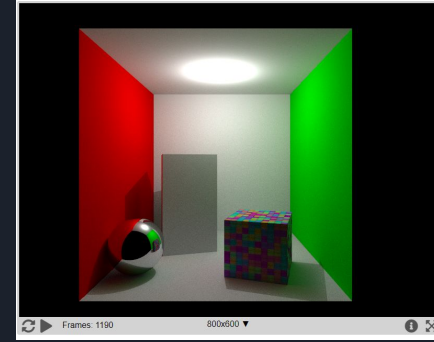
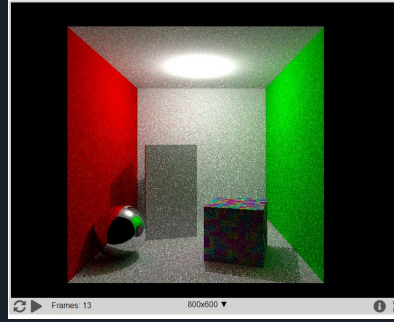


by Agkar

```
Scene Generate Hit Miss Post Process ?
1 void rg_generate ( ) {
2   vec3 At = vec(0.0, 0.0, -2.0);
3   float blending = 1.0 / float(rg_Frame);
4   //if(rg_Mouse.w != -1)
5   {
6     float xAxis = float(rg_Mouse.x) / rg_Canvas.x;
7     float yAxis = float(rg_Mouse.y) / rg_Canvas.y;
8     yAxis = 2.0 * xAxis - 1.0;
9     At.x = -yAxis;
10    At.y = -xAxis;
11    blending = 1.0;
12  }
13  if(rg_Mouse.x == -1)
14    blending = 1.0 / float(rg_Frame);
15  }
16  float ar = float(rg_ScreenHeight) / float(rg_ScreenWidth);
17  float hfovRad = radians(20.0);
18  float vfovRad = 2.0 * atan(tan(hfovRad/2.0)*ar);
19  float hfov = vfovRad;
20  float hfov = hfovRad;
21  vec2 fov = vec2(hfov, vfov);
22  vec3 rendercamview = vec3(0.0, 1);
23  vec3 rendercamup = vec3(0.1, 0);
24  vec3 horizontalAxis = vec3(1.0, 0);
25  vec3 verticalAxis = cross(rendercamview, horizontalAxis);
26  vec3 middle = At + rendercamview;
27  vec3 horizontal = horizontalAxis * tan(fov.x * 0.5);
28  vec3 vertical = verticalAxis * tan(fov.y * 0.5);
29  float sx = (rg_Pixel.x / float(rg_ScreenWidth) + fract(rg_Time) / float(rg_Frame));
30  float sy = (rg_Pixel.y / float(rg_ScreenHeight) + fract(rg_Time) / float(rg_Frame));
31  vec3 pointOnPlaneOnUnitAxisYFromEye = middle +
32    horizontal * ((2.0 * sx) - 1.0) +
33    vertical * ((2.0 * sy) - 1.0);
34  vec3 pointOnImagePlane = At + ((pointOnPlaneOnUnitAxisYFromEye - At));
35  vec3 aperturePoint = At;
36  vec3 apertureToImagePlane = pointOnImagePlane - aperturePoint;
37  vec3 normalToImagePlane = normalize(apertureToImagePlane);
38  rg_RayDirection = vec3(rendercamview, rg_RAY_MAX_DISTANCE);
39  rg_RayOrigin = vec3(At, rg_RAY_ACTIVE_LENGTH);
40  rg_Accumulation = vec3(0.0, 0.0, blending);
41  }
```

# Stages

- 5 customizable stages:
  - Scene, Generate, Hit, Miss and Post-Process



**Figure 2:** High level overview of the Rayground pipeline. Shaded polygons correspond to programmable stages. Rays are submitted as waves and intersection results are provided to the next stages through appropriate API calls.

# Scene

- What objects you want in the scene, ie type
- Where you want them
- Properties
  - Scale
  - Rotation
  - material

Scene Generate Hit Miss Post Process ?

```
1 {
2   "settings": {
3     "depth": 3
4   },
5   "objects": [
6     {
7       "type": "quad",
8       "translate": [ 0, 10.99, 1 ],
9       "scale": [ 0.5, 0.5, 1 ],
10      "rotate": [ 1, 0, 0, 90 ],
11      "material_property0": [ 1, 0, 0 ]
12    },
13    {
14      "type": "quad",
15      "translate": [ 1, 0, 1 ],
16      "scale": [ 2, 2, 1 ],
17      "rotate": [ 0, 1, 0, -90 ],
18      "material_property0": [ 0, 1, 0 ]
19    },
20    {
21      "type": "quad",
22      "translate": [ -1, 0, 1 ],
23      "scale": [ 2, 2, 1 ],
24      "rotate": [ 0, 1, 0, 90 ],
25      "material_property0": [ 1, 0, 0 ]
26    },
27    {
28      "type": "quad",
29      "translate": [ 0, 1, 1 ],
30      "scale": [ 2, 2, 1 ],
31      "rotate": [ 1, 0, 0, 90 ],
32      "material_property0": [ 0.8, 0.8, 0.8 ]
33    },
34    {
35      "type": "quad",
36      "translate": [ 0, -1, 1 ],
37      "scale": [ 2, 2, 1 ],
38      "rotate": [ 1, 0, 0, -90 ],
39      "material_property0": [ 0.8, 0.8, 0.8 ]
40    },
41    {
42      "type": "quad",
43      "translate": [ 0, 0, 2 ],
44      "scale": [ 2, 2, 1 ],
45      "rotate": [ 1, 0, 0, 180 ],
46      "material_property0": [ 1, 1, 1 ]
47    },
48    {
49      "type": "cube",
50      "translate": [ 0.4, -0.77, 0.6 ],
51      "scale": [ 0.55, 0.55, 0.55 ],
52      "rotate": [ 0, 1, 0, 17 ],
53      "material_property0": [ 0.8, 0.8, 0.8 ]
```





# Generate

- Generate primary rays
- Virtual camera
- Spawn rays in parallel for each pixel  
using `rg_generate()`

| <b><i>Generate/Hit/Miss stages</i></b> |                               |                               |
|--|-------------------------------|-------------------------------|
| vec4                                   | <i>rg_[Prev]Accumulation</i>  | ray (prev) accumulation color |
| vec4                                   | <i>rg_[Prev]Payload0</i>      | ray (prev) payload            |
| vec4                                   | <i>rg_[Prev]RayDirection</i>  | ray (prev) direction          |
| vec4                                   | <i>rg_[Prev]RayOrigin</i>     | ray (prev) origin             |
| int                                    | <i>rg_Depth</i>               | ray depth                     |
| bool                                   | <i>rg_TraceOcclusion(...)</i> | ray occlusion query           |
| <b><i>Generate stage</i></b>           |                               |                               |
| void                                   | <i>rg_generate()</i>          | entry point signature         |

# Hit | Miss

- Intersection data is available.
- Calculate distance and angle of primitive to light source
- Spawn new rays
- Miss stage can be used for skyboxes

| <i>Generate/Hit/Miss stages</i> |                               |                               |
|---------------------------------|-------------------------------|-------------------------------|
| vec4                            | <i>rg_[Prev]Accumulation</i>  | ray (prev) accumulation color |
| vec4                            | <i>rg_[Prev]Payload0</i>      | ray (prev) payload            |
| vec4                            | <i>rg_[Prev]RayDirection</i>  | ray (prev) direction          |
| vec4                            | <i>rg_[Prev]RayOrigin</i>     | ray (prev) origin             |
| int                             | <i>rg_Depth</i>               | ray depth                     |
| bool                            | <i>rg_TraceOcclusion(...)</i> | ray occlusion query           |

| <i>Hit stage</i> |                                  |                                   |
|------------------|----------------------------------|-----------------------------------|
| void             | <i>rg_hit()</i>                  | entry point signature             |
| vec3             | <i>rg_Hitpoint</i>               | hit in world space coordinates    |
| vec3             | <i>rg_Normal</i>                 | primitive's geometric normal      |
| int              | <i>rg_MaterialID</i>             | primitive's material ID           |
| vec4             | <i>rg_MaterialPropertyI(...)</i> | material properties, $I = [0, 7]$ |

| <i>Miss stage</i> |                  |                       |
|-------------------|------------------|-----------------------|
| void              | <i>rg_miss()</i> | entry point signature |

*Post-Process stage*





# Post Process

- Functions almost like a fragment shader
- Used for filtering, tone mapping, etc
- De-noising

| <i>Post Process stage</i> |                             |                               |
|---------------------------|-----------------------------|-------------------------------|
| void                      | <i>rg_post_process()</i>    | entry point signature         |
| vec4                      | <i>rg_PixelColor</i>        | final pixel colour            |
| <i>rg_Image2D</i>         | <i>rg_AccumulatedImage</i>  | 2D accumulation image handle  |
| vec4                      | <i>rg_ImageFetch2D(...)</i> | retrieve texels from 2D image |



# Conclusion

- Evaluation of 20 students
  - 80% - very positive effect of learning ray tracing
  - 73% - wanted to experiment further
- Web-based graphics has drawbacks
  - Bi-directional methods
  - Animation systems



Demo?

<https://rayground.com/view/dExaQa67tqI>