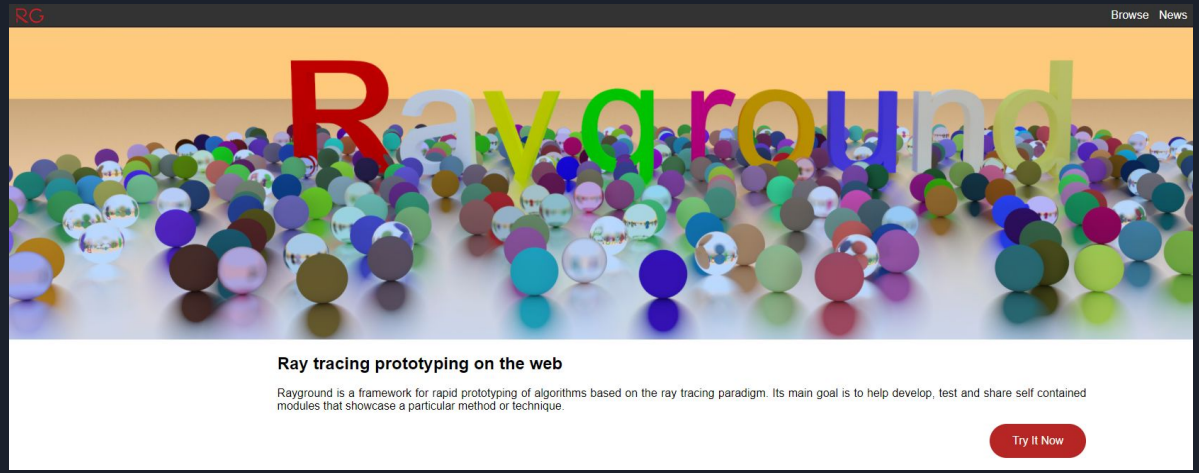
Rayground:

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

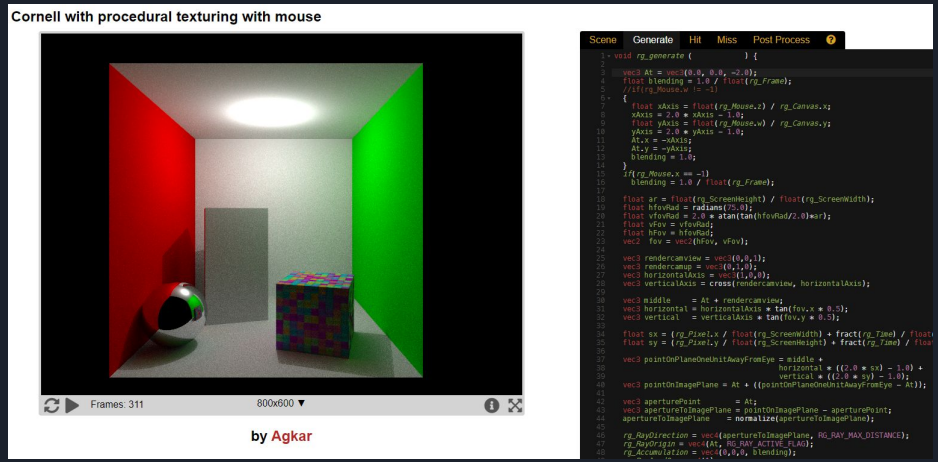


Why

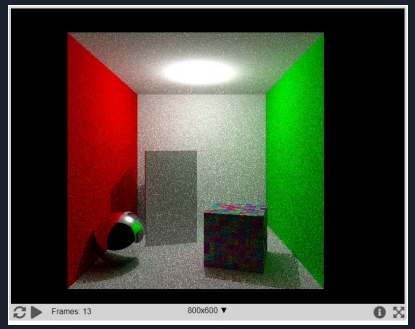
- To learn ray tracing without distractions
  - The specific characteristics of a framework/ API, etc
- Learning ray tracing in OpenGL/c++ has a lot of additional challenges
- Low-level APIs like OptiX can be quite technical
- Argues that learning about ray tracing should be more accessible
  - Webapps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

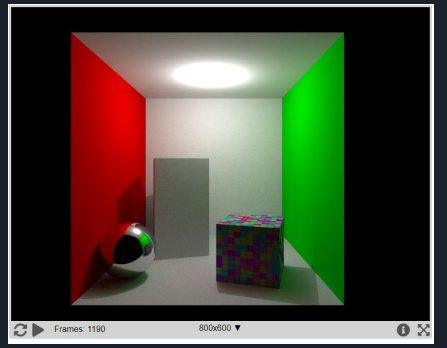
How

- 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()



Stages





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

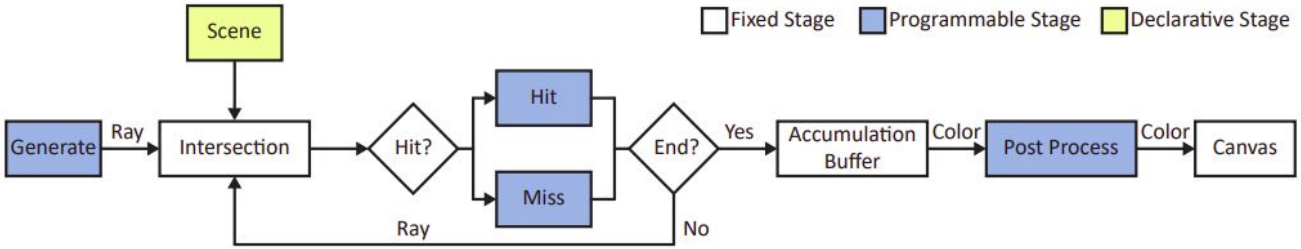


Figure 2: *High level overview of the Rayground pipeline. Shaded polygons correspond to programmable stages. Rays are submitted as waves and intersection results are provided to the next stages through appropriate API calls.*

Scene

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

Scene | Generate | Hit | Miss | Post Process

```
{
  "settings": {
    "depth": 3
  },
  "objects": [
    {
      "type": "quad",
      "translate": [ 0, 10.99, 1 ],
      "scale": [ 0.5, 0.5, 1 ],
      "rotate": [ 1, 0, 0, 90 ],
      "material_property0": [ 1, 0, 0 ]
    },
    {
      "type": "quad",
      "translate": [ 1, 0, 1 ],
      "scale": [ 2, 2, 1 ],
      "rotate": [ 0, 1, 0, -90 ],
      "material_property0": [ 0, 1, 0 ]
    },
    {
      "type": "quad",
      "translate": [ -1, 0, 1 ],
      "scale": [ 2, 2, 1 ],
      "rotate": [ 0, 1, 0, 90 ],
      "material_property0": [ 1, 0, 0 ]
    },
    {
      "type": "quad",
      "translate": [ 0, 1, 1 ],
      "scale": [ 2, 2, 1 ],
      "rotate": [ 1, 0, 0, 90 ],
      "material_property0": [ 0.8, 0.8, 0.8 ]
    },
    {
      "type": "quad",
      "translate": [ 0, -1, 1 ],
      "scale": [ 2, 2, 1 ],
      "rotate": [ 1, 0, 0, -90 ],
      "material_property0": [ 0.8, 0.8, 0.8 ]
    },
    {
      "type": "quad",
      "translate": [ 0, 0, 2 ],
      "scale": [ 2, 2, 1 ],
      "rotate": [ 1, 0, 0, 180 ],
      "material_property0": [ 1, 1, 1 ]
    },
    {
      "type": "cube",
      "translate": [ 0.4, -0.77, 0.6 ],
      "scale": [ 0.55, 0.55, 0.55 ],
      "rotate": [ 0, 1, 0, 17 ],
      "material_property0": [ 0.8, 0.8, 0.8 ]
```

Generate

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

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

Hit | Miss

- Intersection data is available.
- Calculate distance and angle of primitive to light source
- Spawn new rays

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

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

- Miss stage can be used for skyboxes

Miss stage		
void	*rg_miss()*	entry point signature

Post Process stage

Post Process

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

Post Process stage		
void	*rg_post_process()*	entry point signature
vec4	*rg_PixelColor*	final pixel colour
rg_Image2D	*rg_AccumulatedImage*	2D accumulation image handle
vec4	*rg_ImageFetch2D(...)*	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