WebGL with three.js

ENEI 2016

José Ferrão

deti

a vida académica na prática AAUAv

NEECT

NEI

Setup

Text editor (sublime, emacs, vim, brackets, notepad, ...)

Web Browser (Firefox, chrome, edge, ...)

Phyton 3.0+

Files

Files for the workshop available at

github.com/tentone/enei2016

Presentation

José Manuel Miranda Ferrão

MIECT, Aveiro University

tentone@outlook.com

github.com/tentone

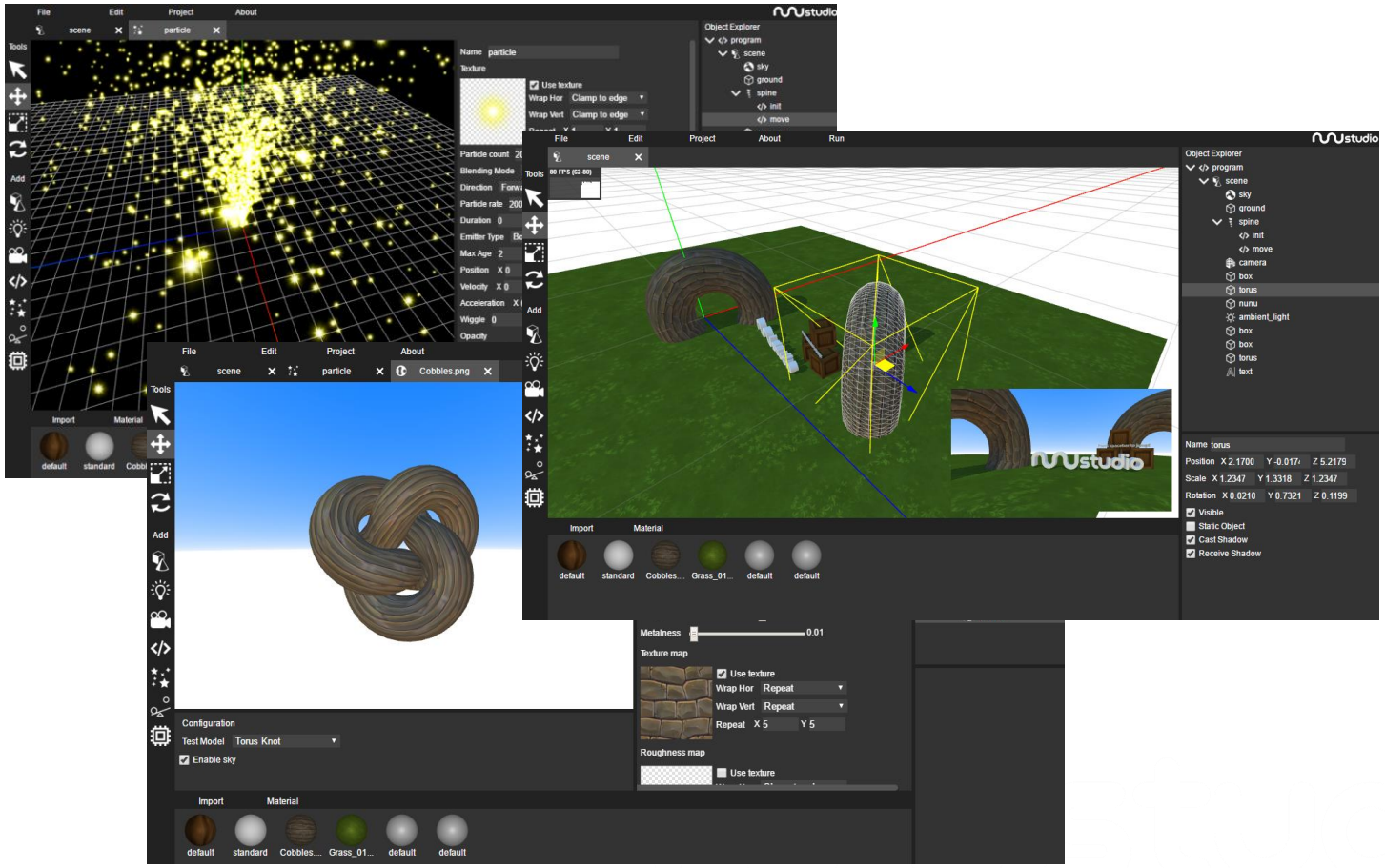
twitter.com/tentonej

IDKwGL (github.com/tentone/idkwgl)

idkGL

- WebGL based library for 3D graphics
- My first contact with WebGL

nunuStudio (github.com/tentone/nunuStudio)



- IDE for web 3D and VR applications development
- Based on three.js

NIM GAME

CALC RUSH

DANGER!

FOURTH DIMENSION

in use

Objectives

- Introduce ~~base~~ concepts about graphic programming
- Explore the three.js library
- Introduce WebVR with three.js

Javascript

- Originally developed by Brendan Eich in 1995
- JavaScript is standardized by the ECMAScript specifications
- A browser reads the code and runs it directly

- Javascript is getting faster
- But still ... stop using it for everything ... please

Javascript

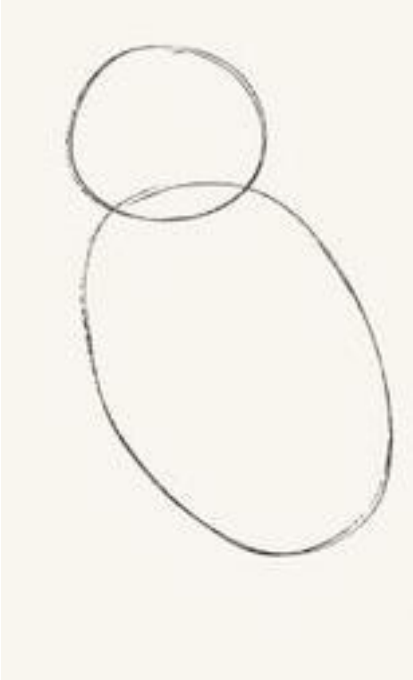
WebGL

- WebGL is a Javascript API for rendering 3D graphics using the computer GPU
- No plugins are required
- WebGL specification is close the to the OpenGL ES 2.0 specification

- Graphics programming can be hard
  - Mathematics
  - GLSL

How to draw an Owl

"A fun and create guide for beginners"

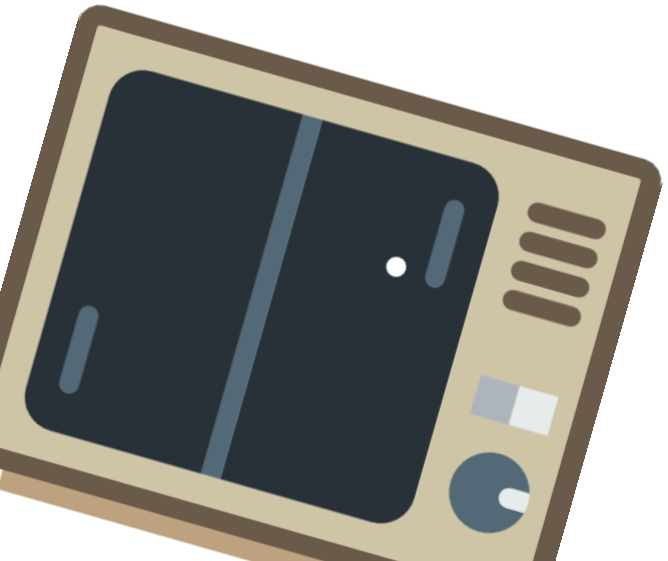


First step
Draw 2 circles

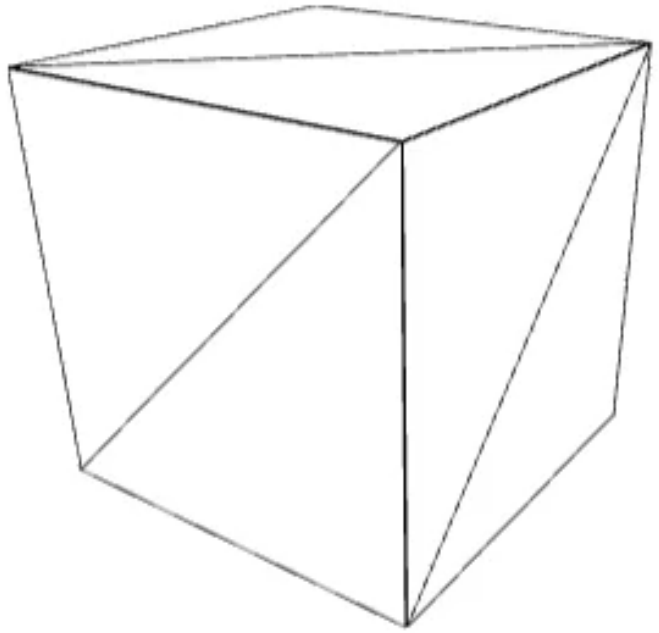
Second step
Draw the rest of the damn Owl

Rendering pipeline

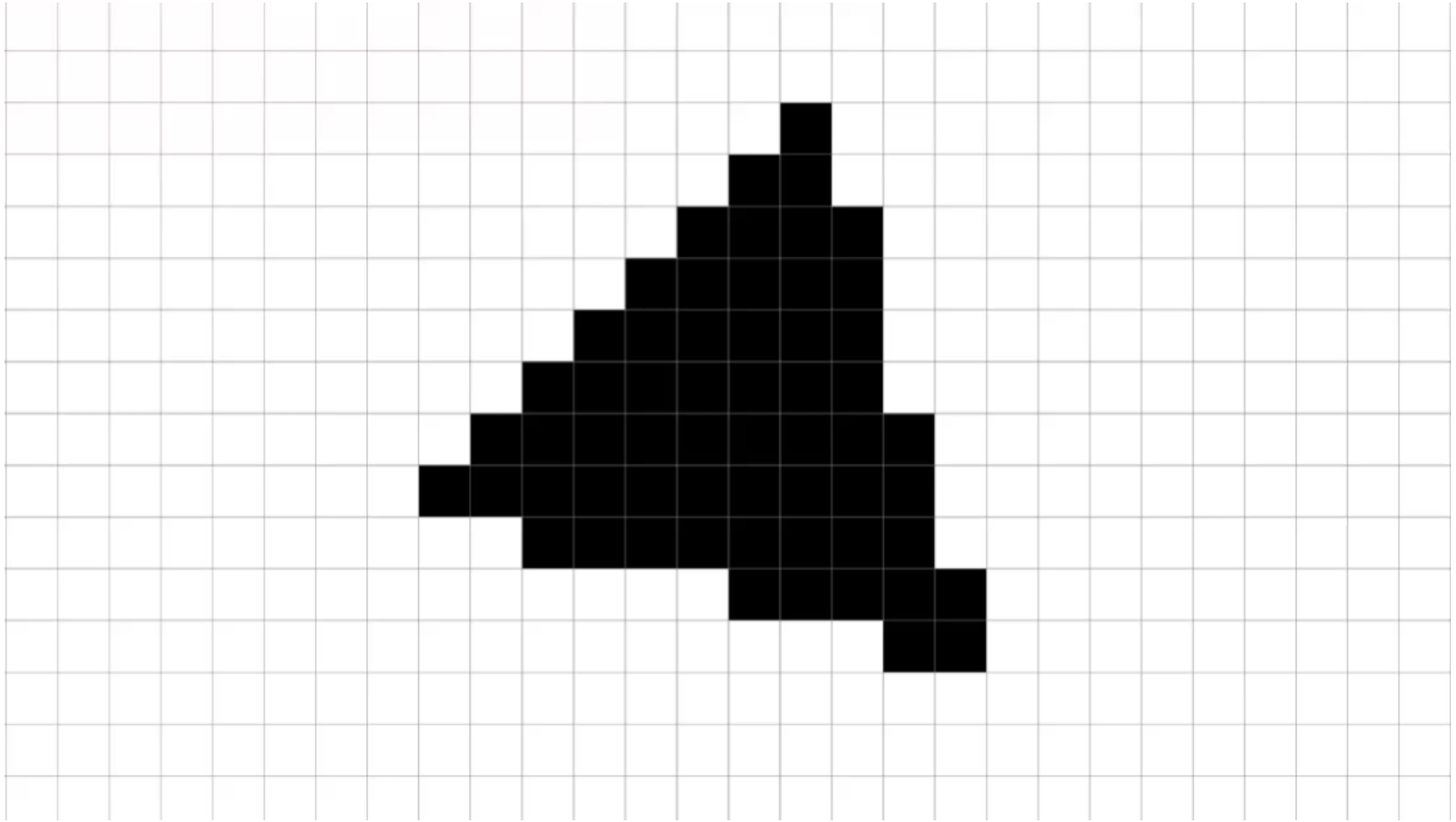
"From data to screen"



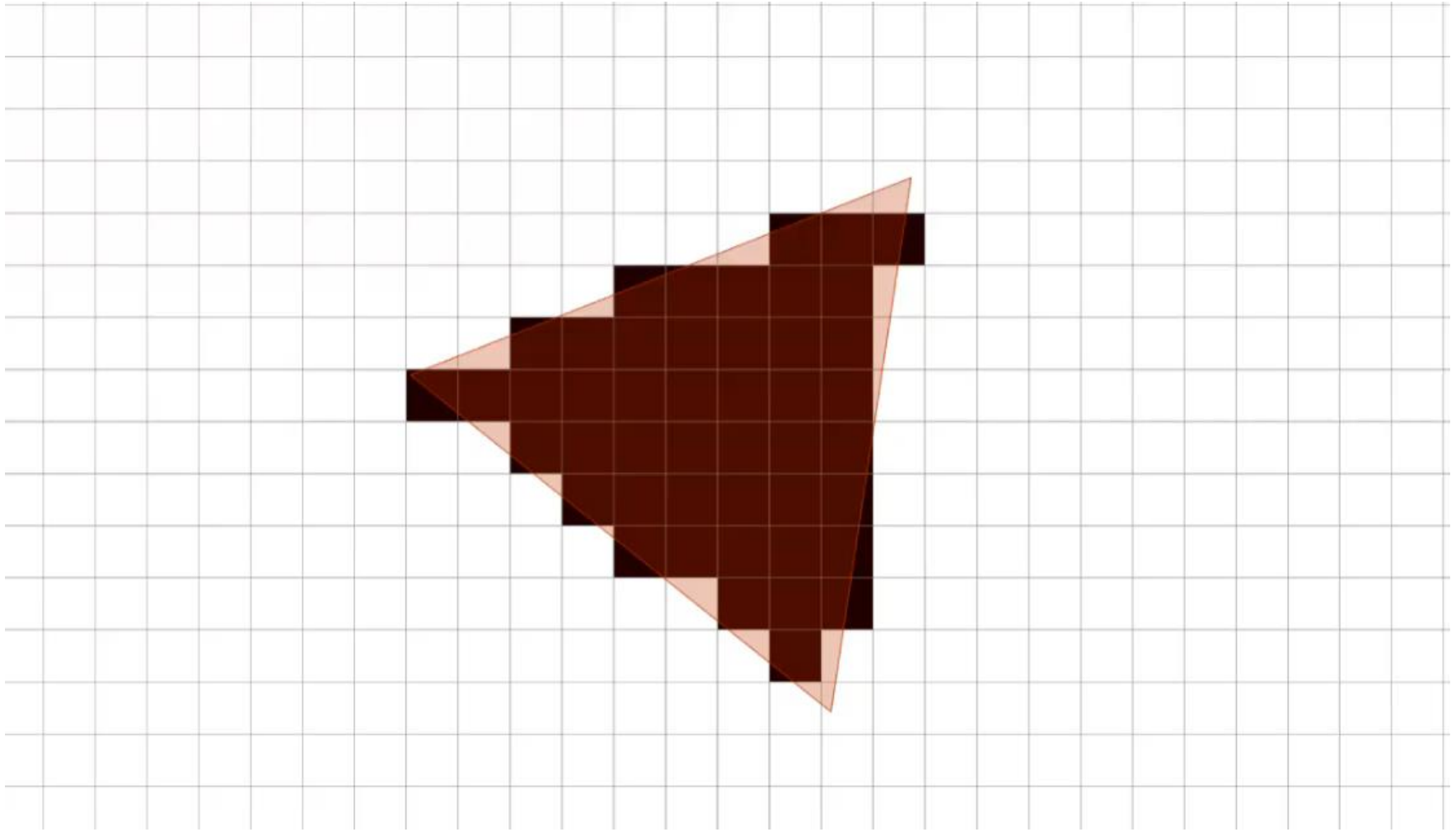
Triangles



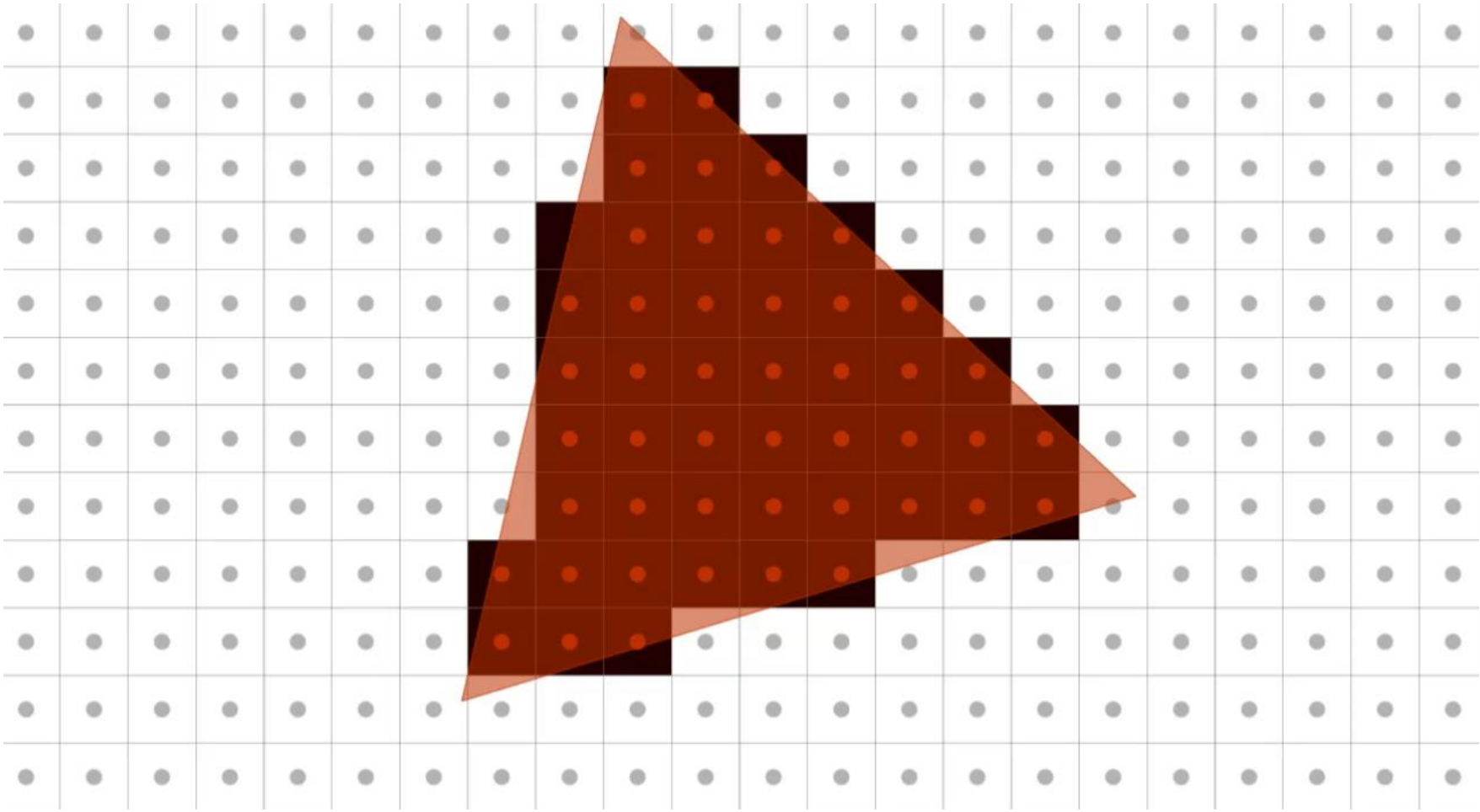
Pixel graphics



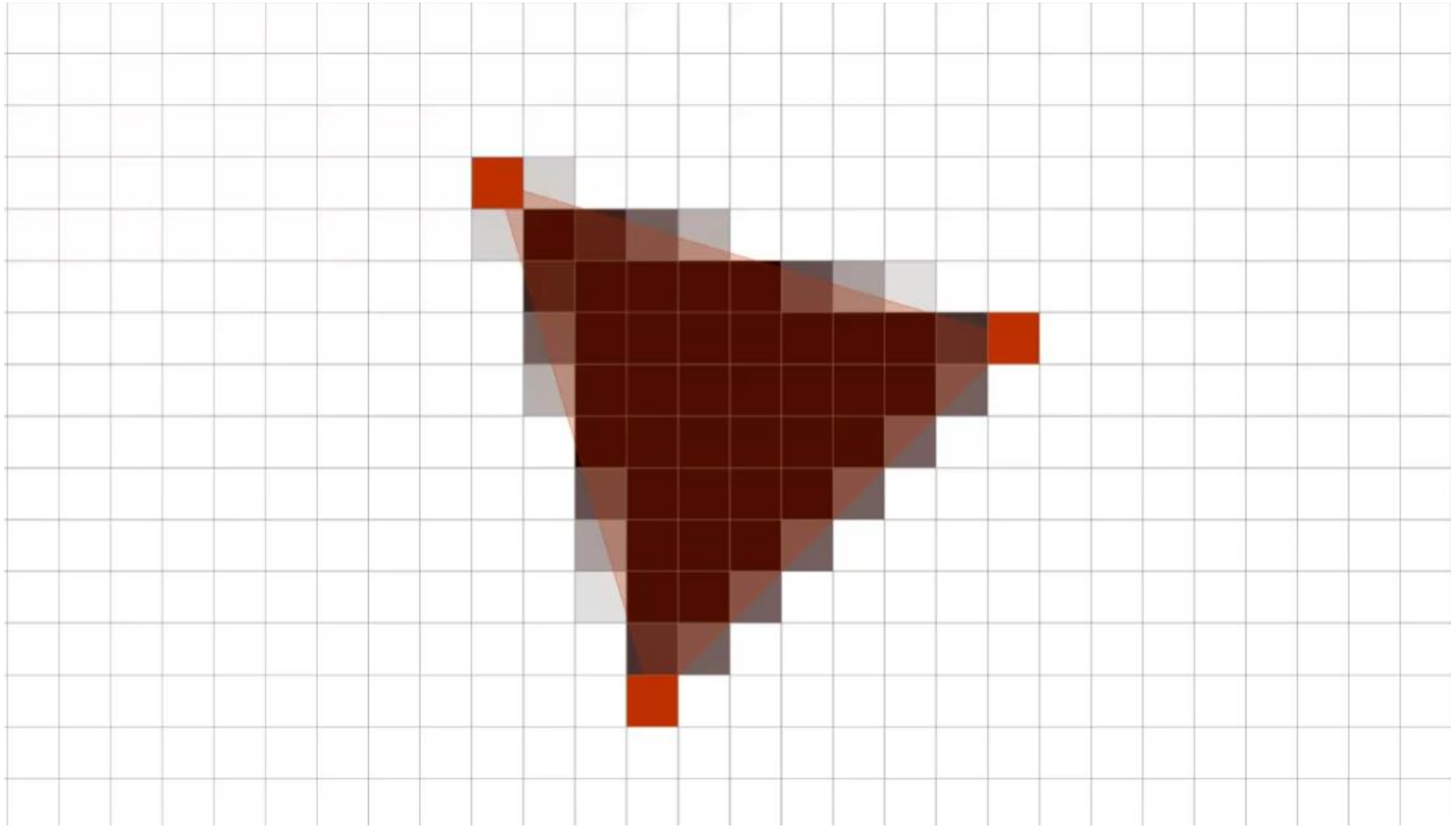
Sub-pixel accuracy



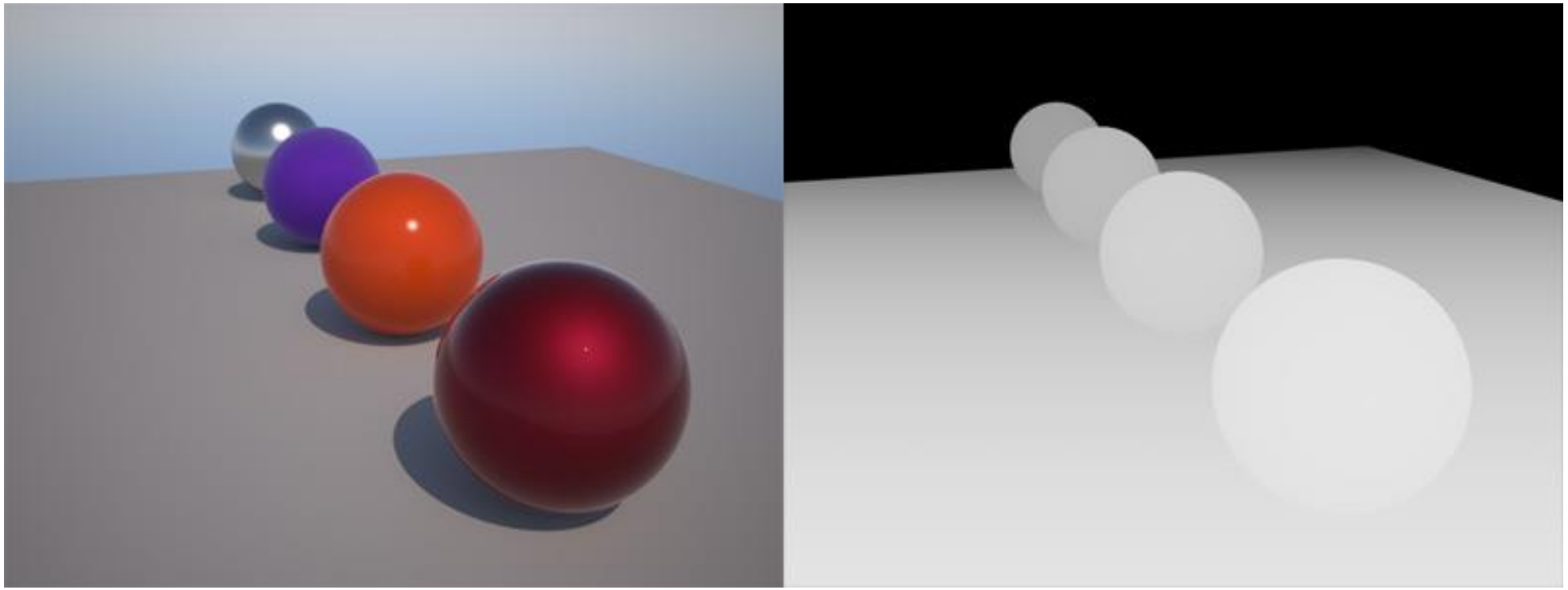
Sub-pixel accuracy



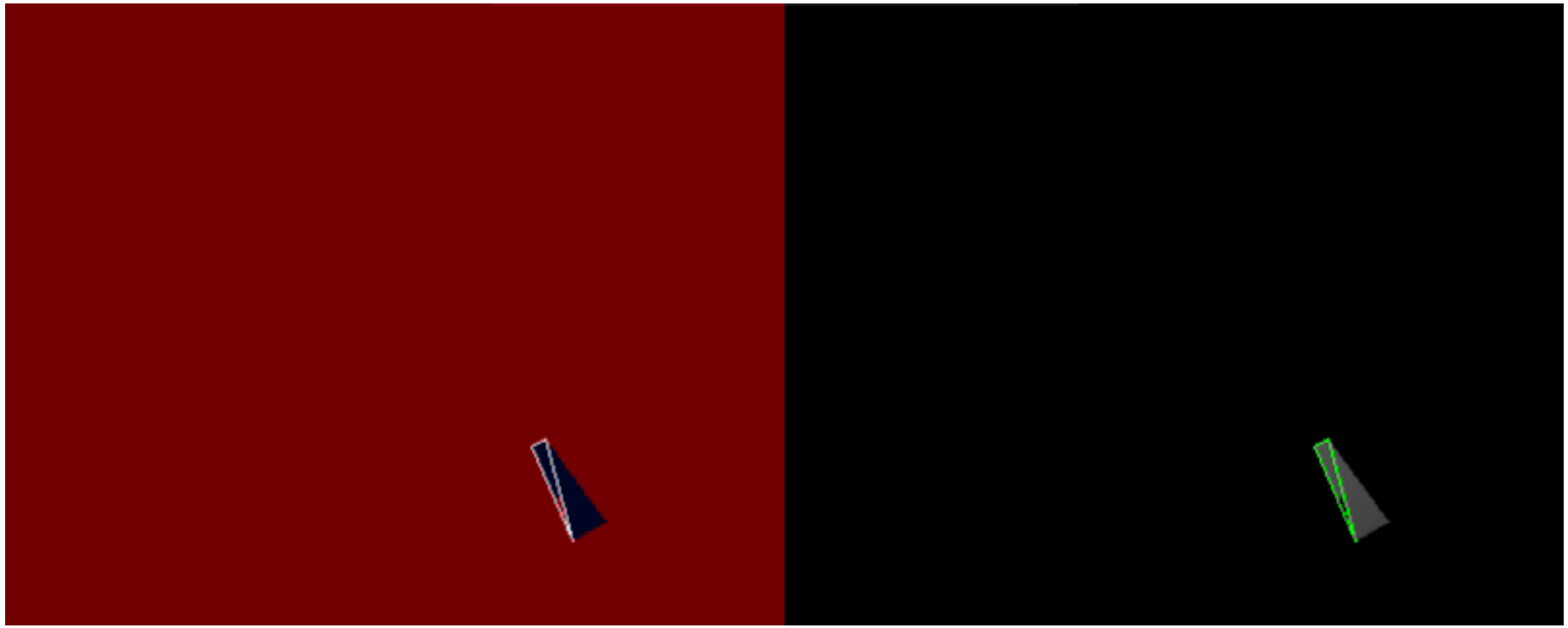
Anti-Aliasing



Depth Buffer

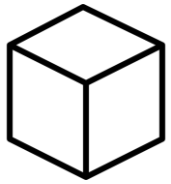


Depth Buffer (http://orbides.org/apps/superslow.html)



GLSL

GLSL is a C-like language that runs in the GPU to draw images



Vertex shader

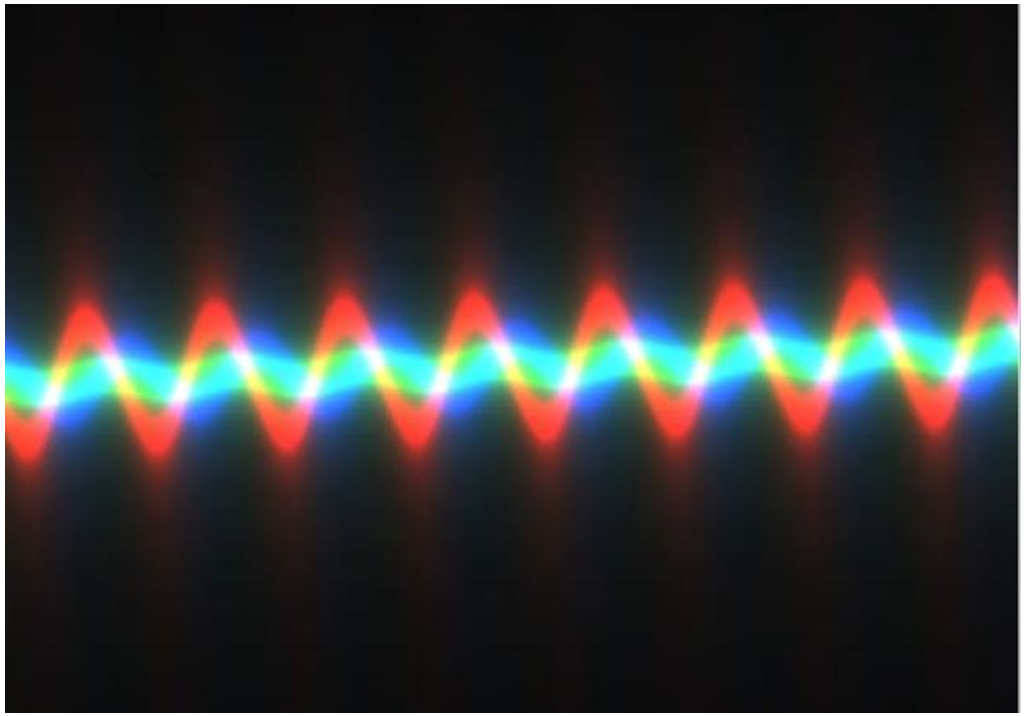
Operate on the vertex level and can be used to deform objects

Fragment shader

Operate at pixel level and are used to control how the objects surface is draw

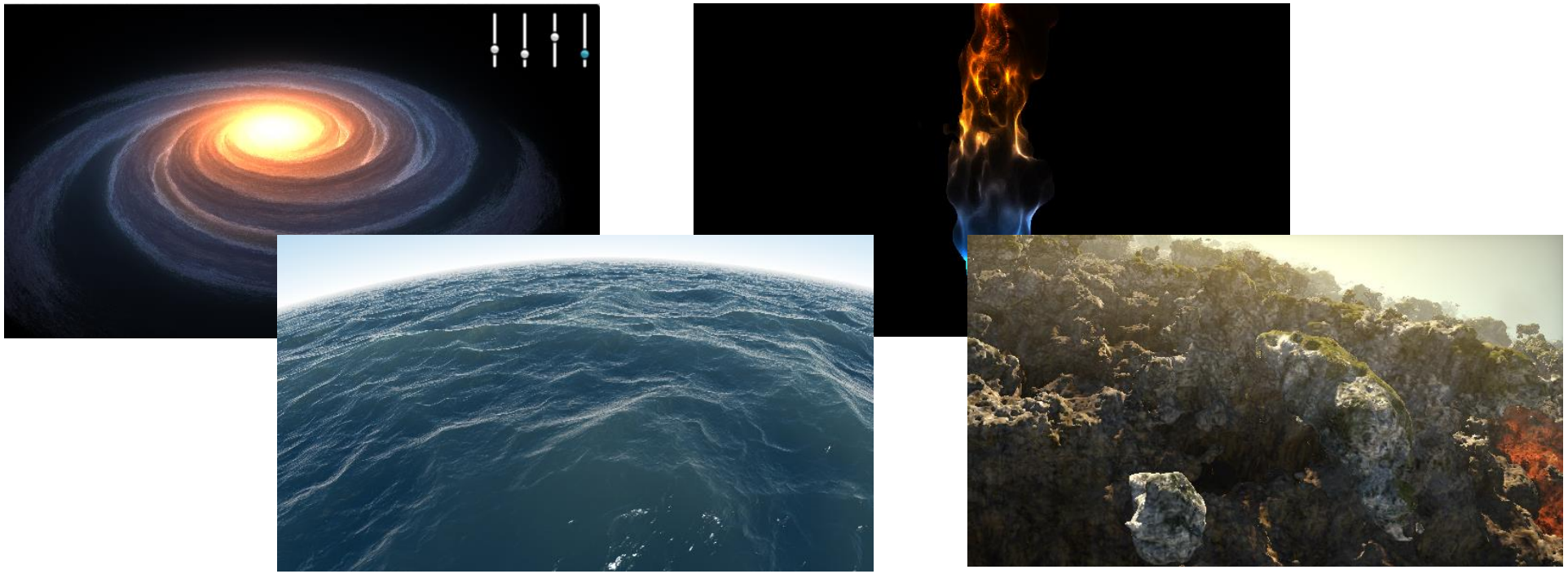
GLSL

```
uniform float time;
uniform vec2 resolution;

float wiggle(float x, float y, float alpha)
{
    return 1.0 / distance(50.0 * y, cos(time) * 5.0
        * alpha * sin(x * 50.0 + cos(time) + time*alpha));
}

void main()
{
    vec2 pos = gl_FragCoord.xy / resolution - 0.5;

    float theta = 0.3 * sin(time * 0.4);
    mat2 rot = mat2 (cos(theta), -sin(theta),sin(theta), cos(theta));
    pos = rot * pos;

    float red = wiggle(pos.x, pos.y, 2.0);
    float green = wiggle(pos.x, pos.y, 0.2);
    float blue = wiggle(pos.x, pos.y, 0.9);

    gl_FragColor = vec4(vec3(red, green, blue), 2.0);
}
```

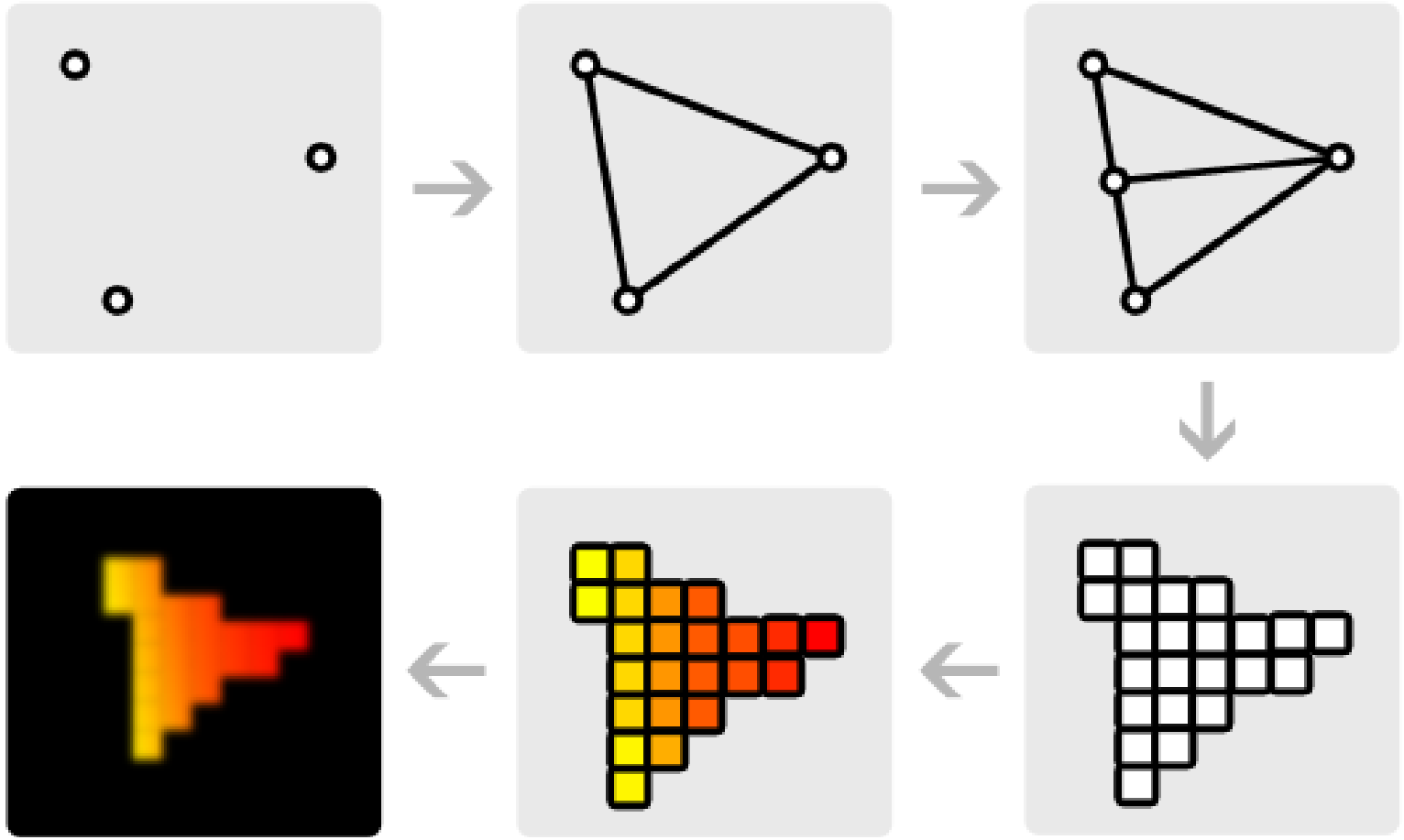


Shadertoy (www.shadertoy.com)

Website to share GLSL shaders



Rendering pipeline



three.js

three.js (www.threejs.org)

- Cross-browser high level library for WebGL graphics
- User all over the internet
- Compreensive API for beginners

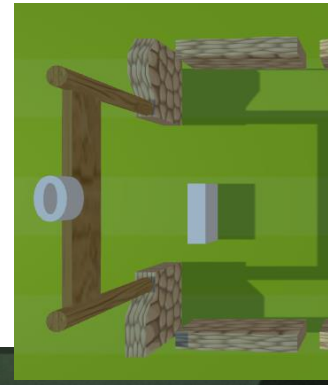
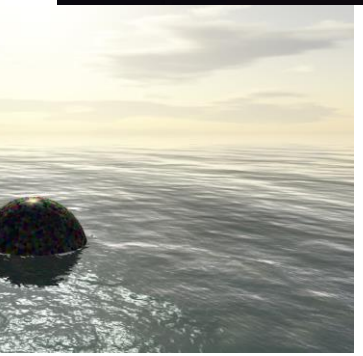
Released by Ricardo Cabello to GitHub in 2010

github.com/mrdoob

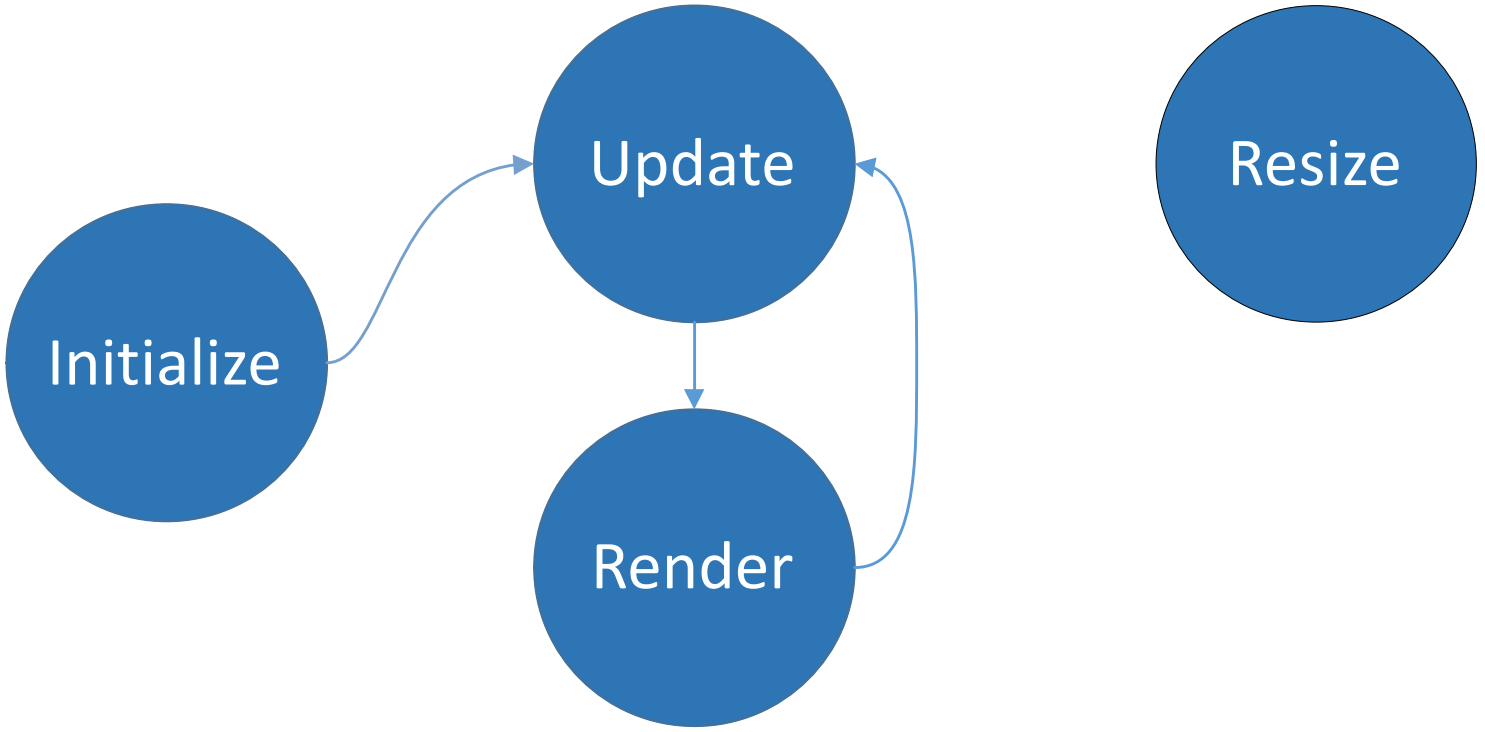
twitter.com/mrdoob

three.js




Lifecycle

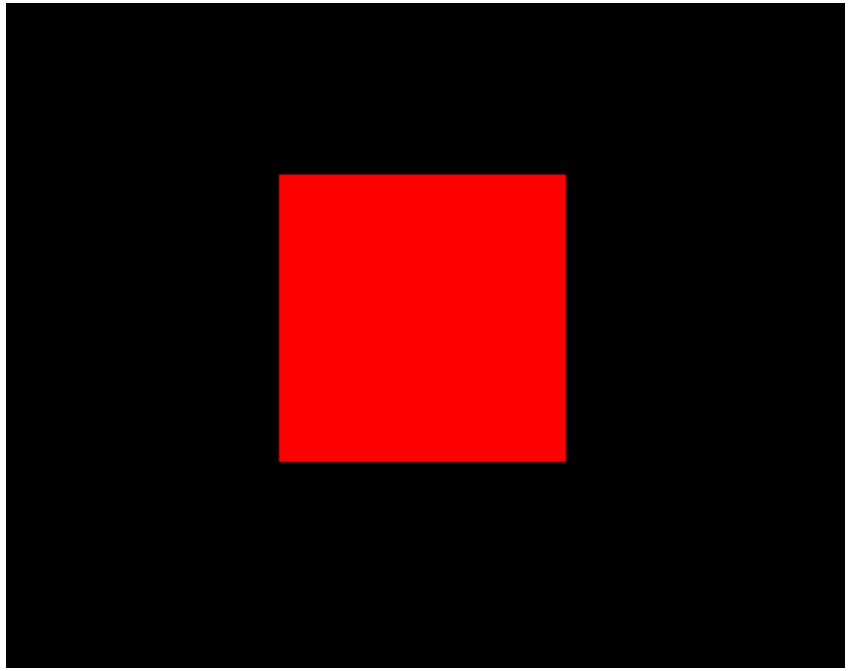


Lifecycle

```
//Initialization code here

update();

function update()
{
    //Update code here
}

function resize()
{
    //Resize code here
}
```

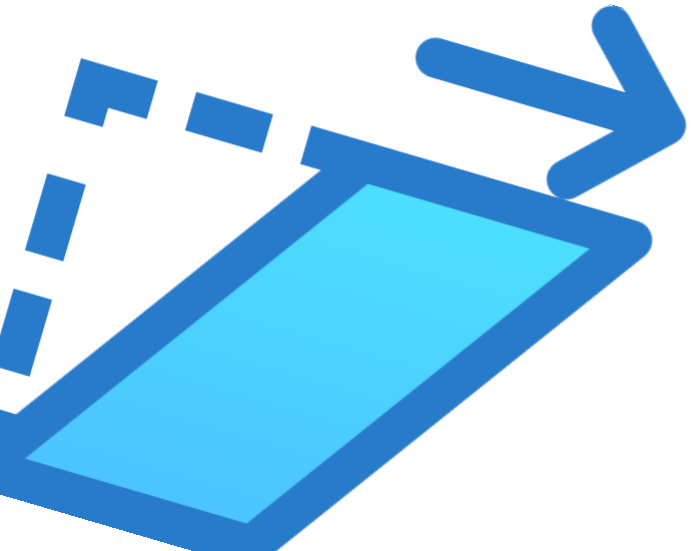
Hello world

```
//WebGl renderer
var renderer = new THREE.WebGLRenderer({canvas: canvas, antialias: true});
renderer.setPixelRatio(window.devicePixelRatio);
renderer.setSize(canvas.width, canvas.height);

//Scene
var scene = new THREE.Scene();

//Camera
var camera = new THREE.PerspectiveCamera(60, canvas.width/canvas.height, 0.1, 1000);
camera.position.set(0, 0, 3);
scene.add(camera);

//Material (defines how the object surface in draw)
var material = new THREE.MeshBasicMaterial({color: "#FF0000"});

//Geometry (defines the object form)
var geometry = new THREE.BoxGeometry(1, 1, 1);

//Mesh (combines a geometry and a material)
var cube = new THREE.Mesh(geometry, material);
scene.add(cube);
```

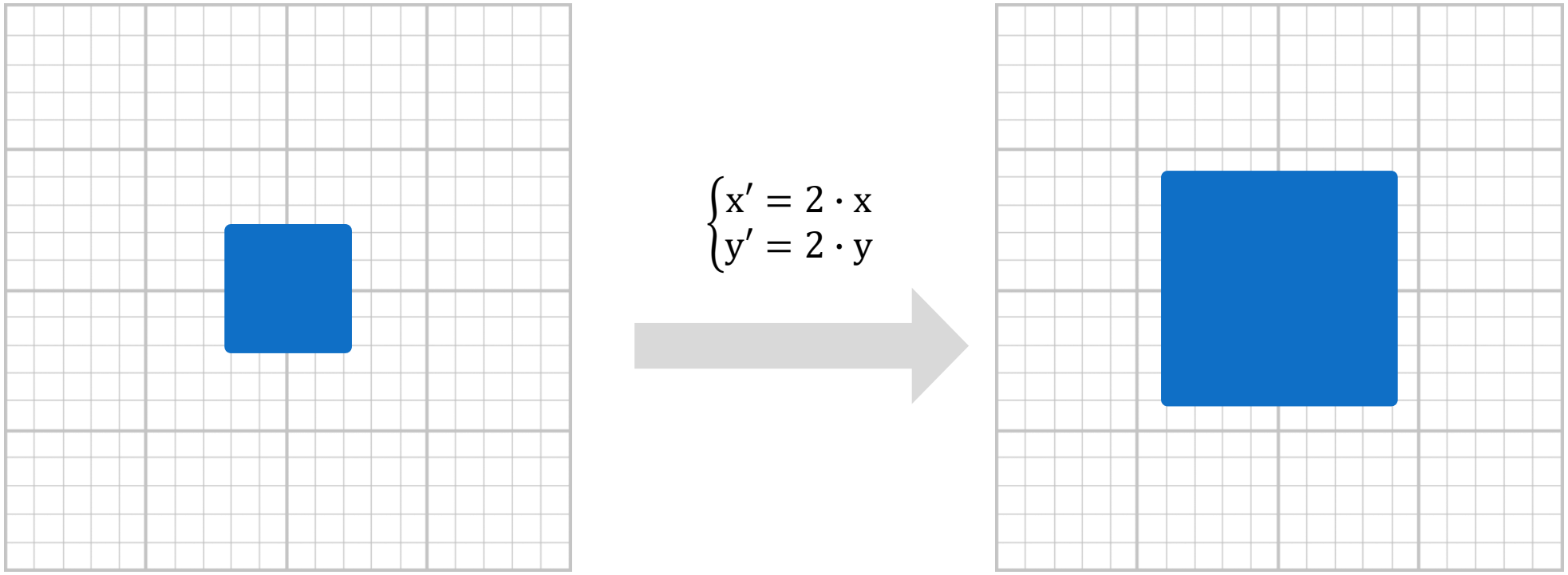


Can we make it move?

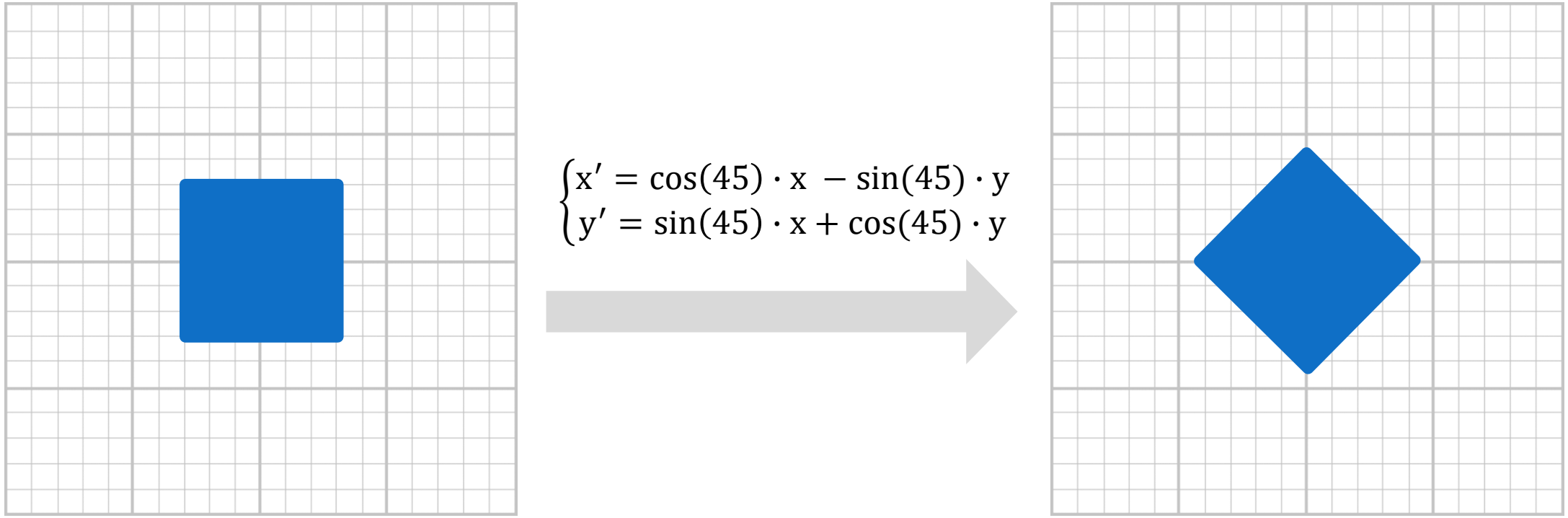
Transformations



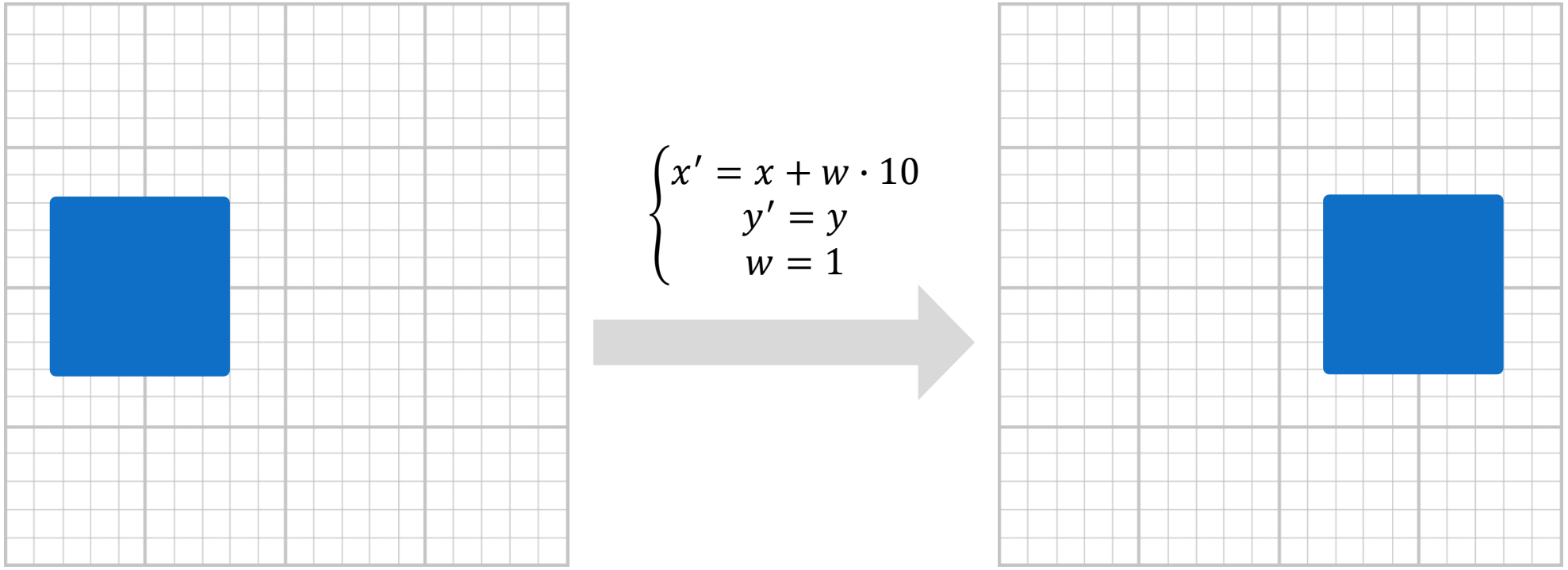
Scale



Rotation


$\begin{cases} x' = \cos(45) \cdot x - \sin(45) \cdot y \\ y' = \sin(45) \cdot x + \cos(45) \cdot y \end{cases}$

Translation



Matrix Transformations

- Most of the graphic API's apply linear transformations using transformation matrixes
- A transformation matrix is multiplied by the model vertices to get the transformated vertices

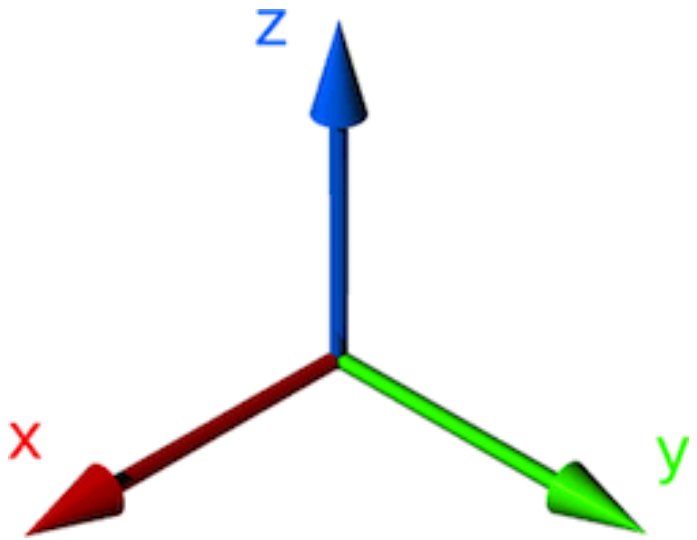
$$\begin{cases} x' = a \cdot x + b \cdot y + c \cdot z + d \cdot w \\ y' = e \cdot x + f \cdot y + g \cdot z + h \cdot w \\ z' = i \cdot x + j \cdot y + k \cdot z + m \cdot w \\ w' = n \cdot x + o \cdot y + p \cdot z + q \cdot w \end{cases}$$

$$[x' \quad y' \quad z' \quad w'] = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & m \\ n & o & p & q \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3D Objects

- Objects are represented in a 3D space
  - Position
  - Scale
  - Rotation

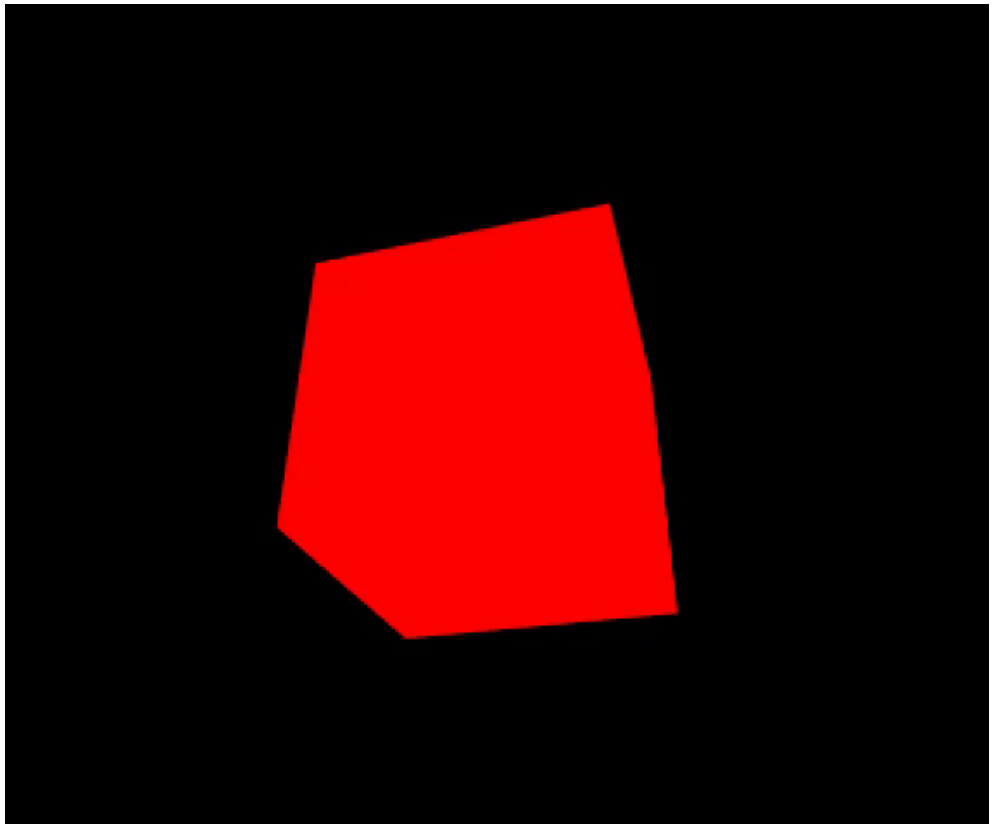
```
(...)
cube.rotation.y = 3.14;

cube.position.set(0, 0, 0);
cube.position.x = 2;

cube.scale.set(1, 1, 1)
(...)
```



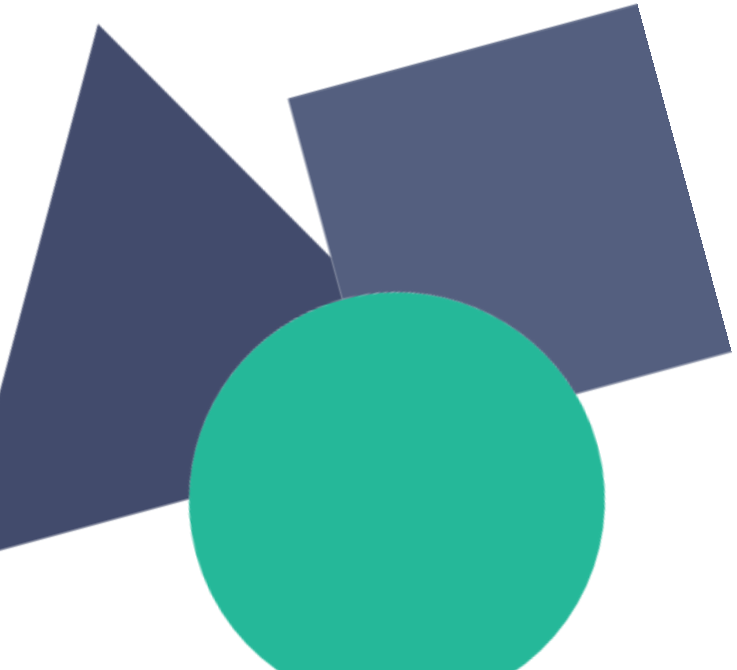
- Objects are organized in a tree structure
- Parents properties affect the children objects

Hello world!

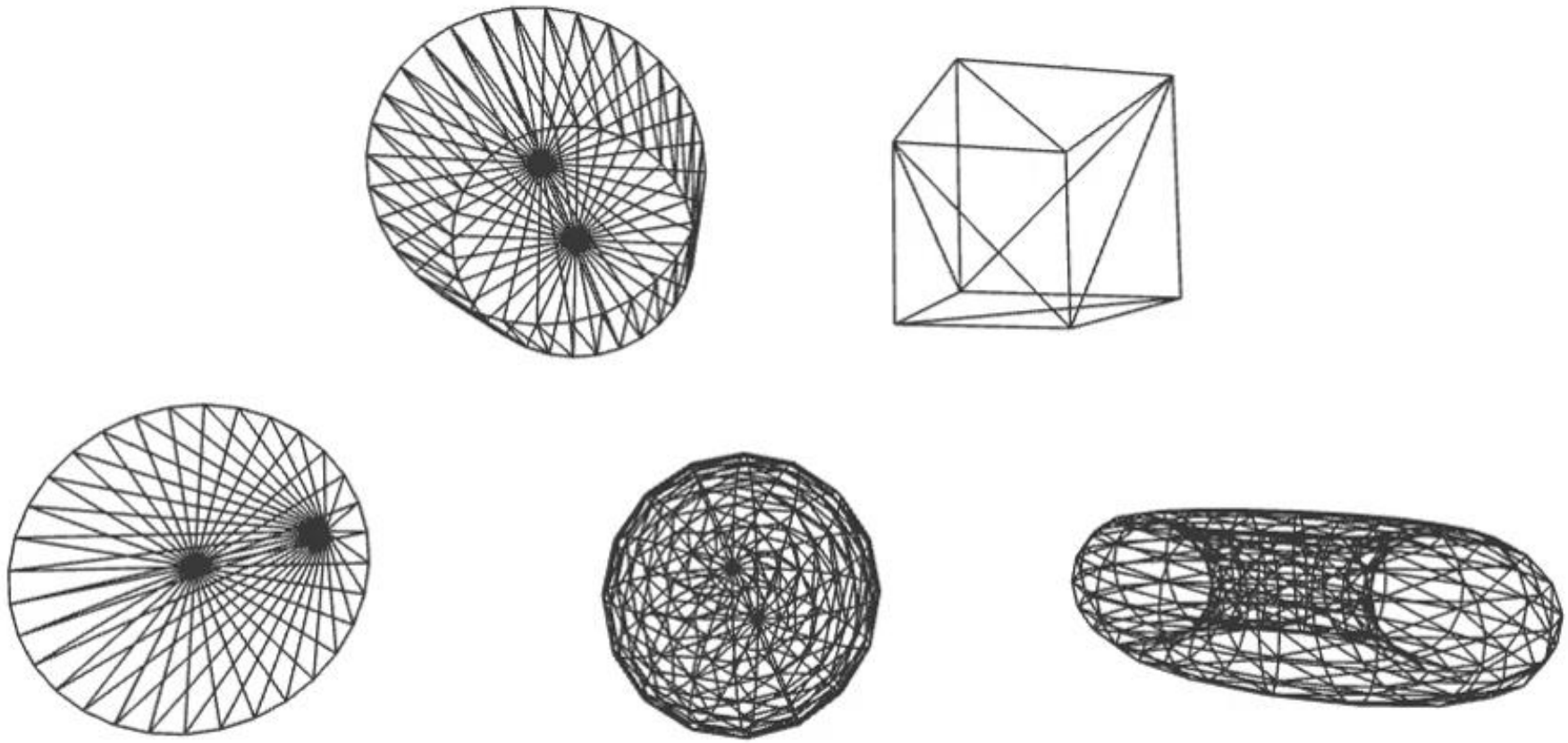
```
//Rotate box
cube.rotation.y += 0.01;
cube.rotation.x += 0.02;
```



Geometries



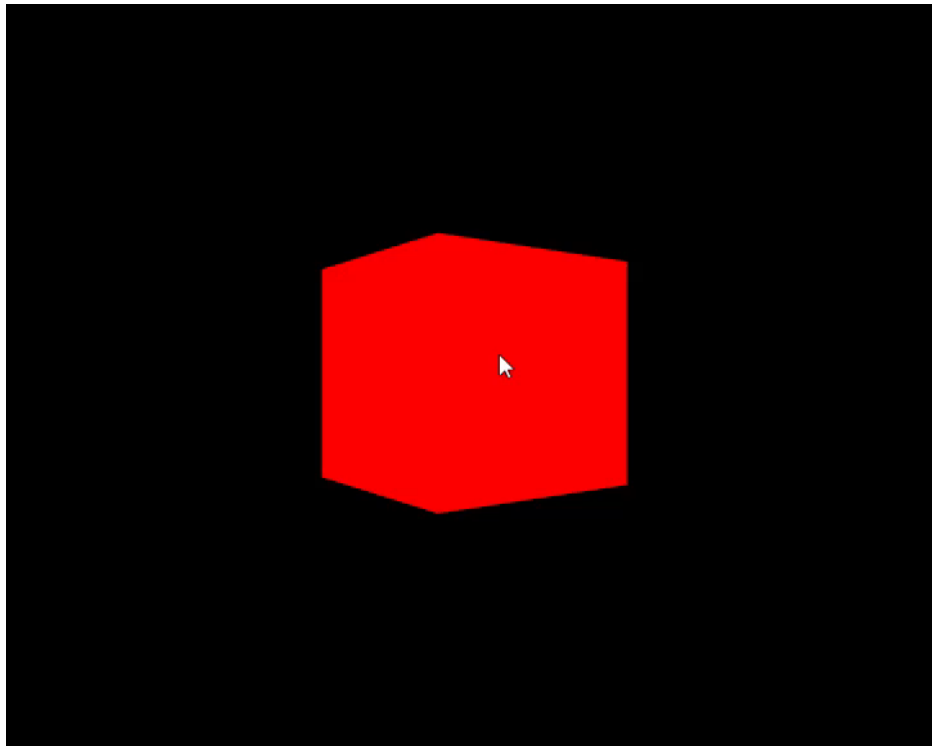
Geometries



Input

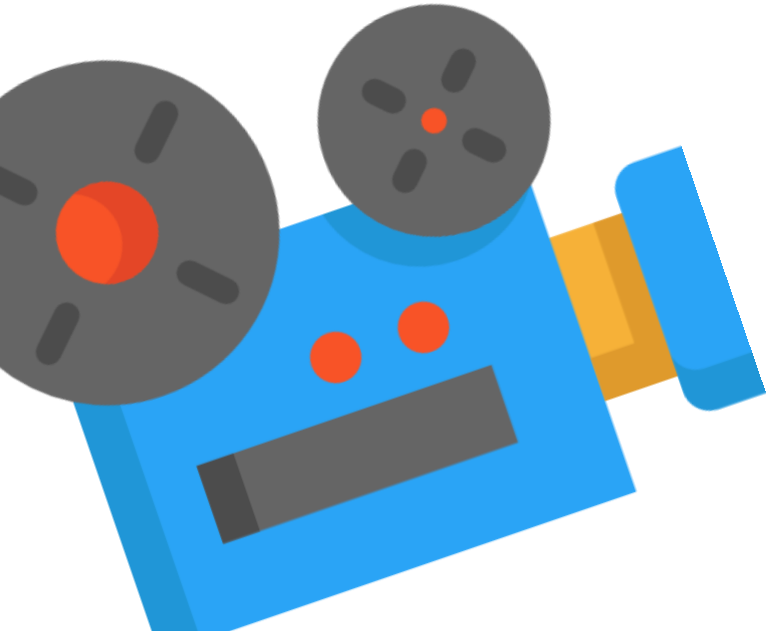
Mouse and keyboard input

```
//Create keyboard and mouse input objects
var keyboard = new Keyboard();
var mouse = new Mouse();

(...)

//Before rendering
keyboard.update();
mouse.update();

if(keyboard.keyPressed(Keyboard.LEFT))
{
    cube.position.x -= 0.1;
}

if(keyboard.keyPressed(Keyboard.RIGHT))
{
    cube.position.x += 0.1;
}

cube.rotation.y += mouse.delta.x / 200;
```

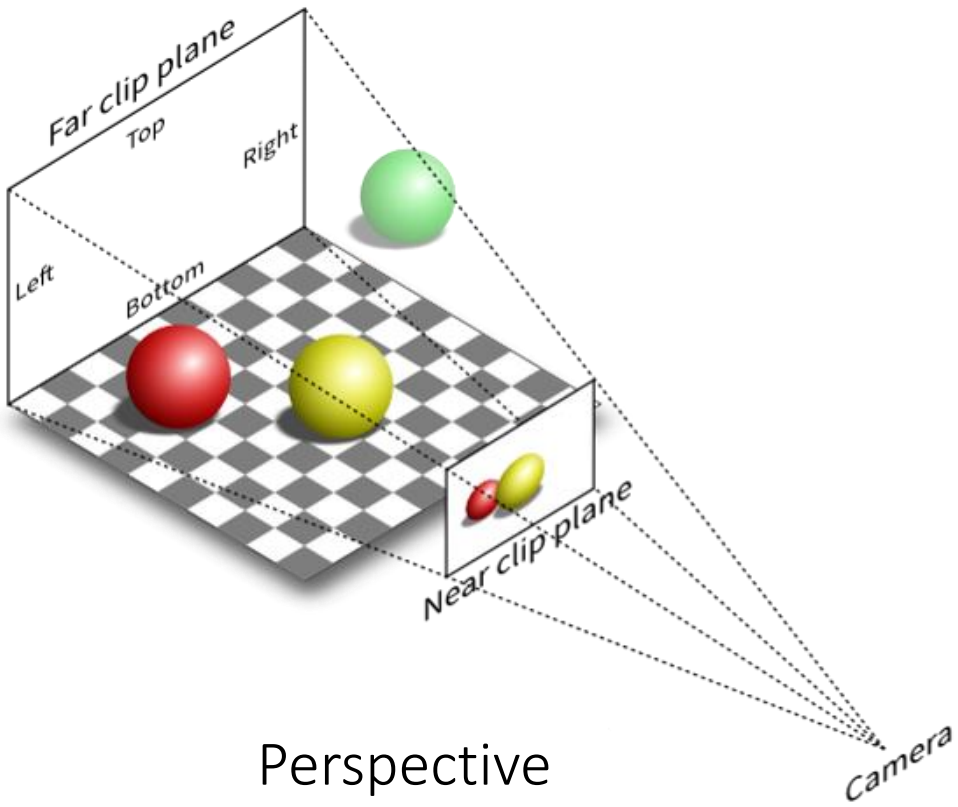


Sync Input

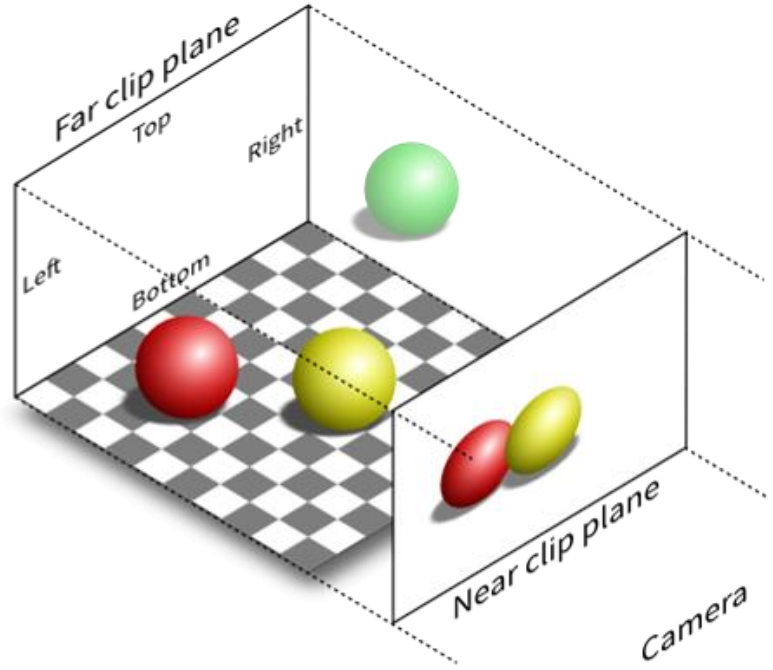
Cameras



Cameras

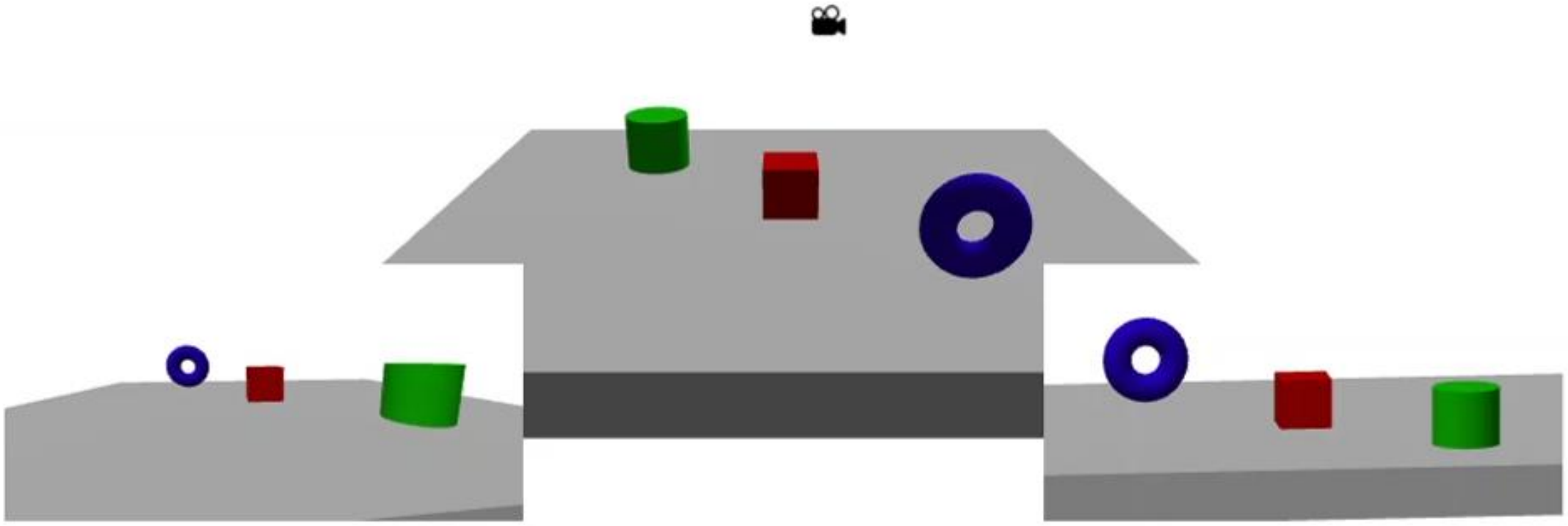


Perspective



Orthographic

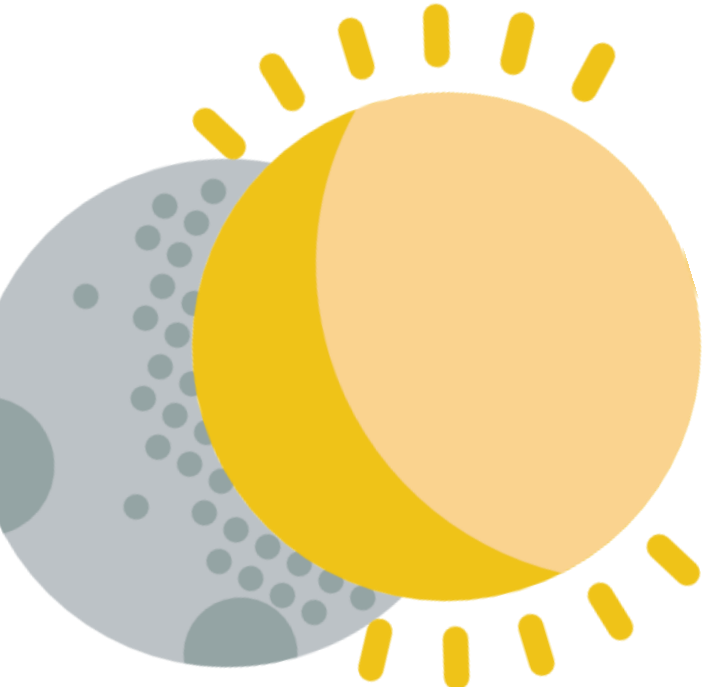
Cameras



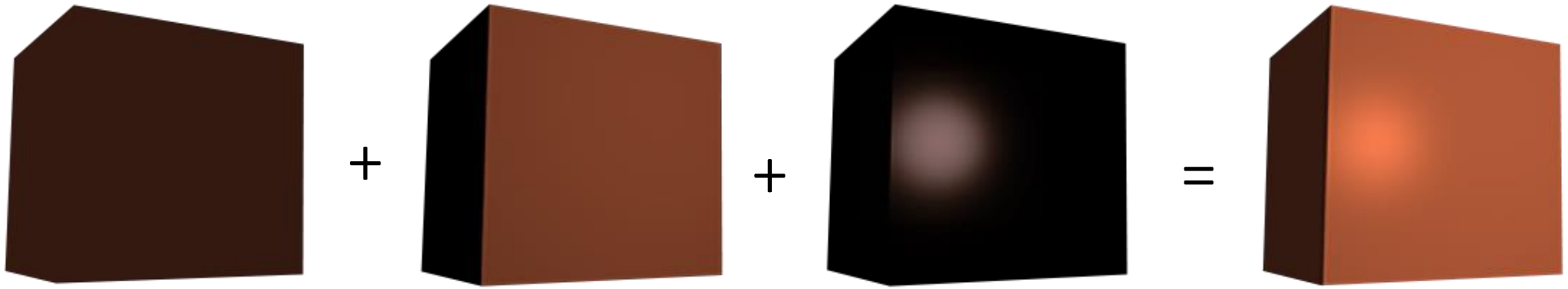
Perspective

Orthographic

Lights

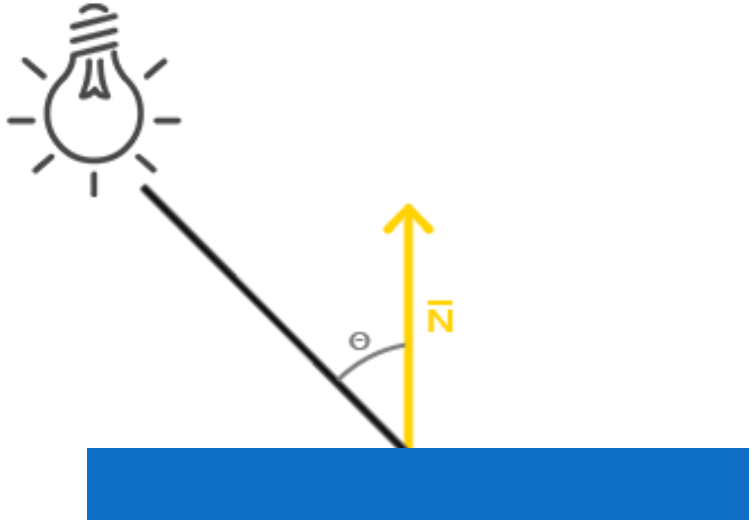


Lights

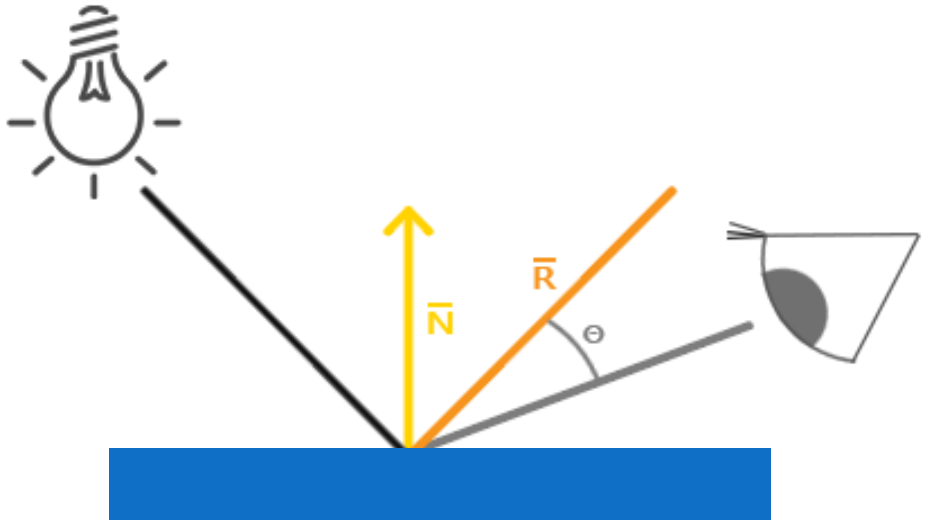


Ambient Diffuse Specular Combined

Lights



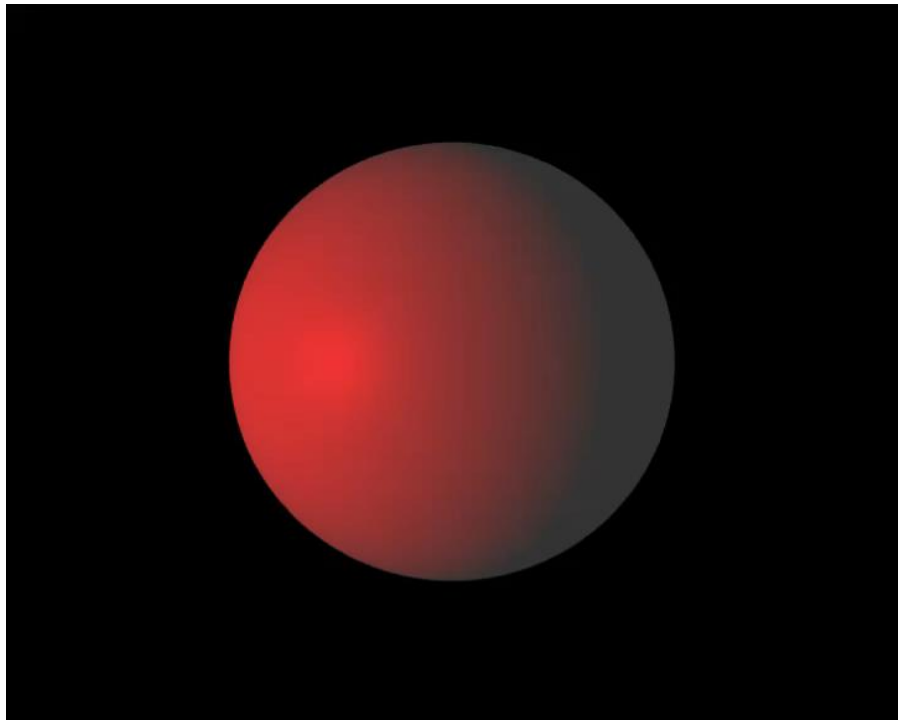
Diffuse



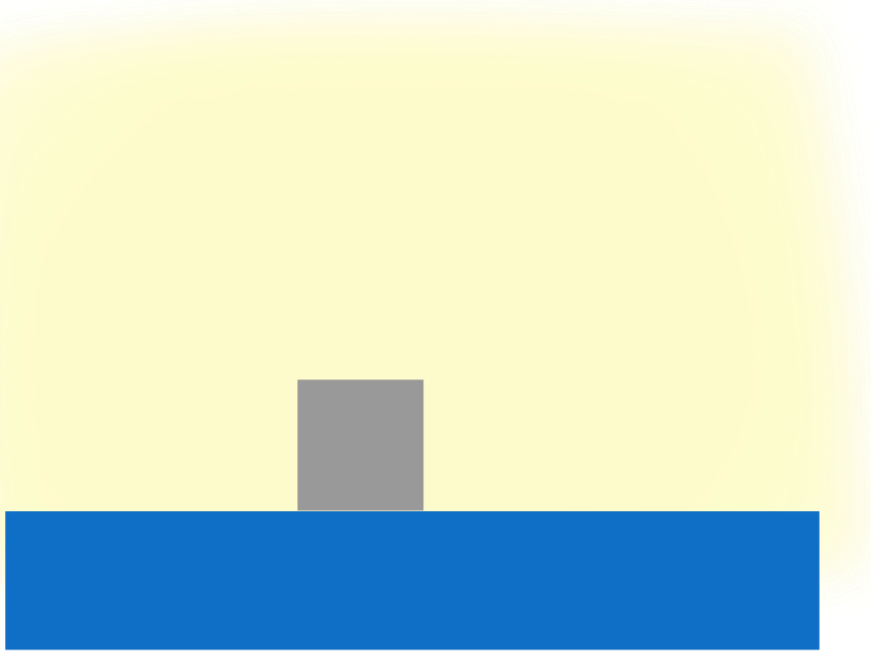
Specular

Lights

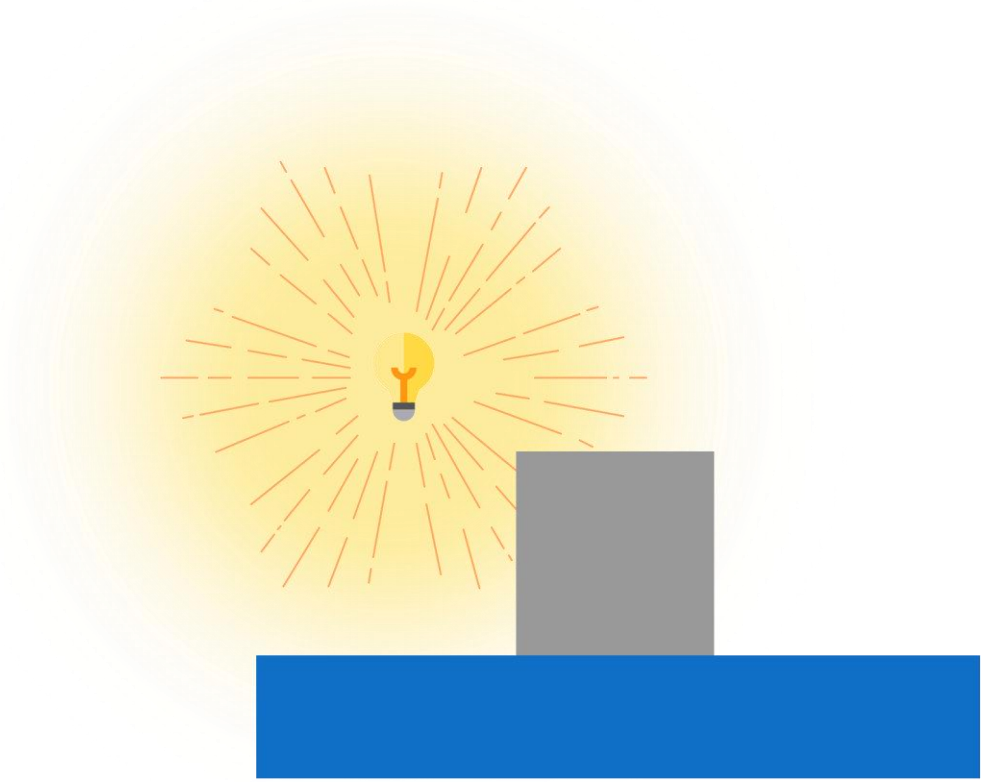
```
//Sphere
var material = new THREE.MeshPhongMaterial();
(...)

//Light
var light = new THREE.PointLight();
light.color = new THREE.Color("#AA0000");
light.position.set(0, 0, 3);
scene.add(light);

//Ambient light
var ambient = new THREE.AmbientLight();
ambient.color = new THREE.Color("#333333");
scene.add(ambient);
```

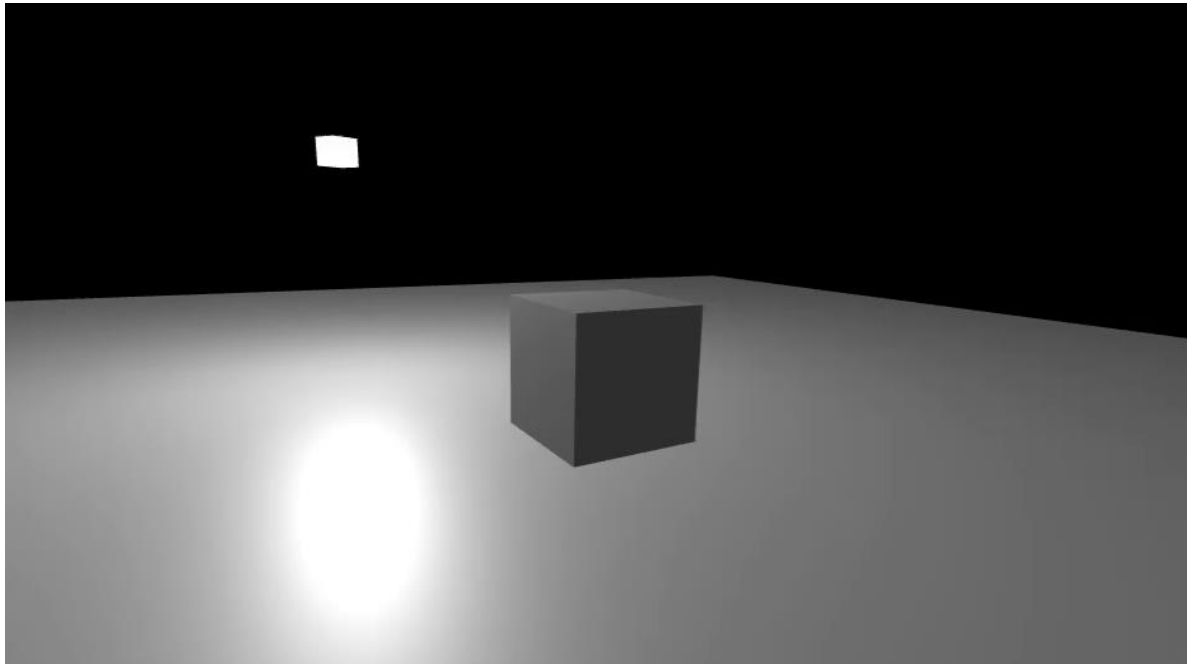


Ambient Lights

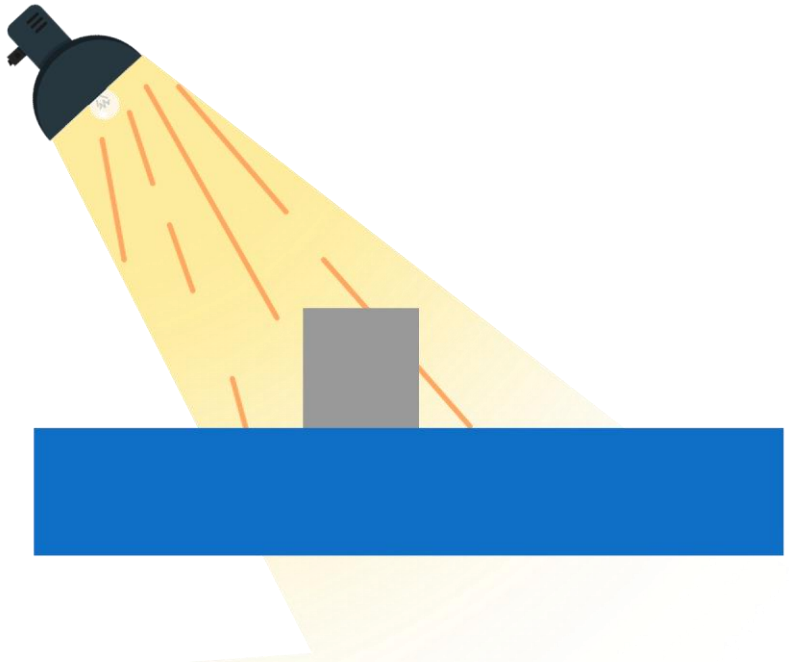


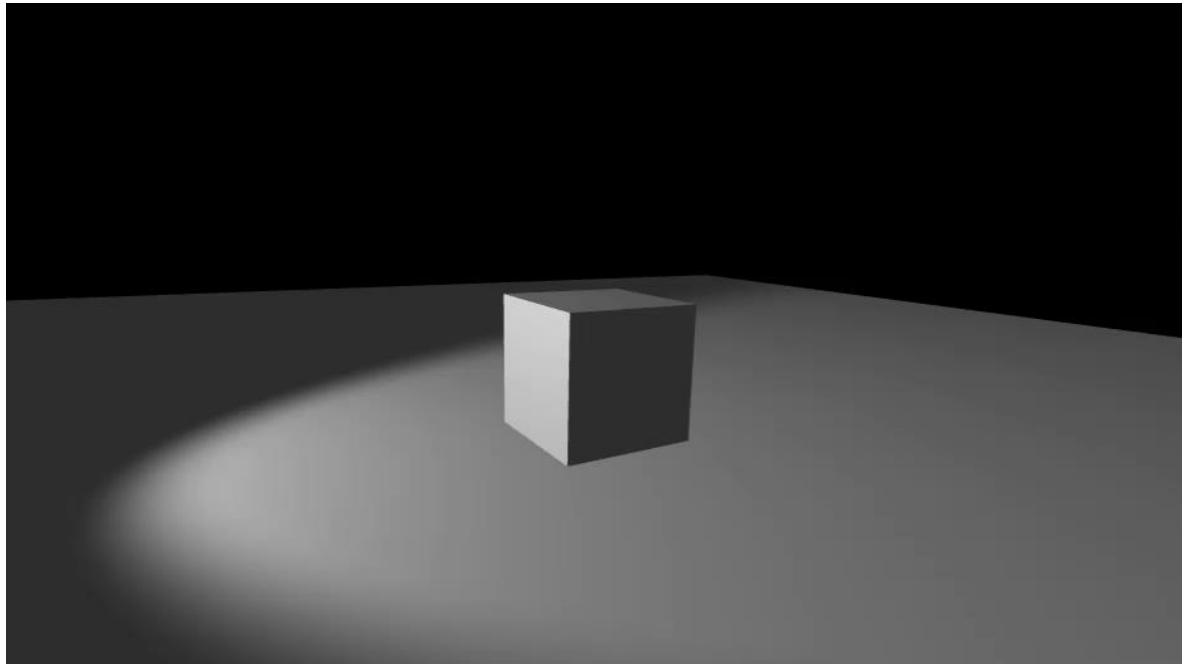
Point Lights



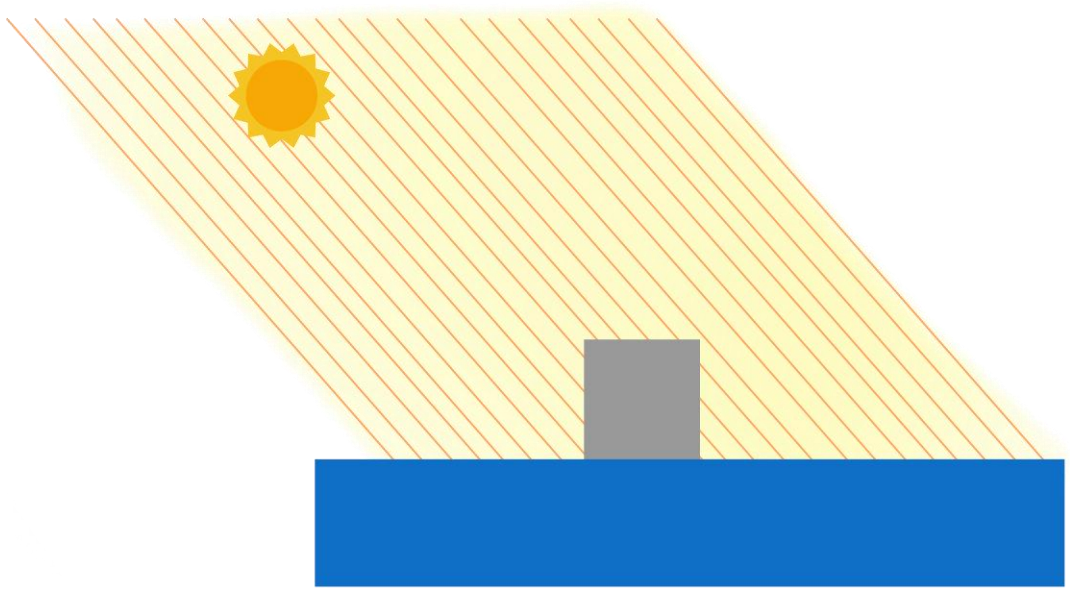


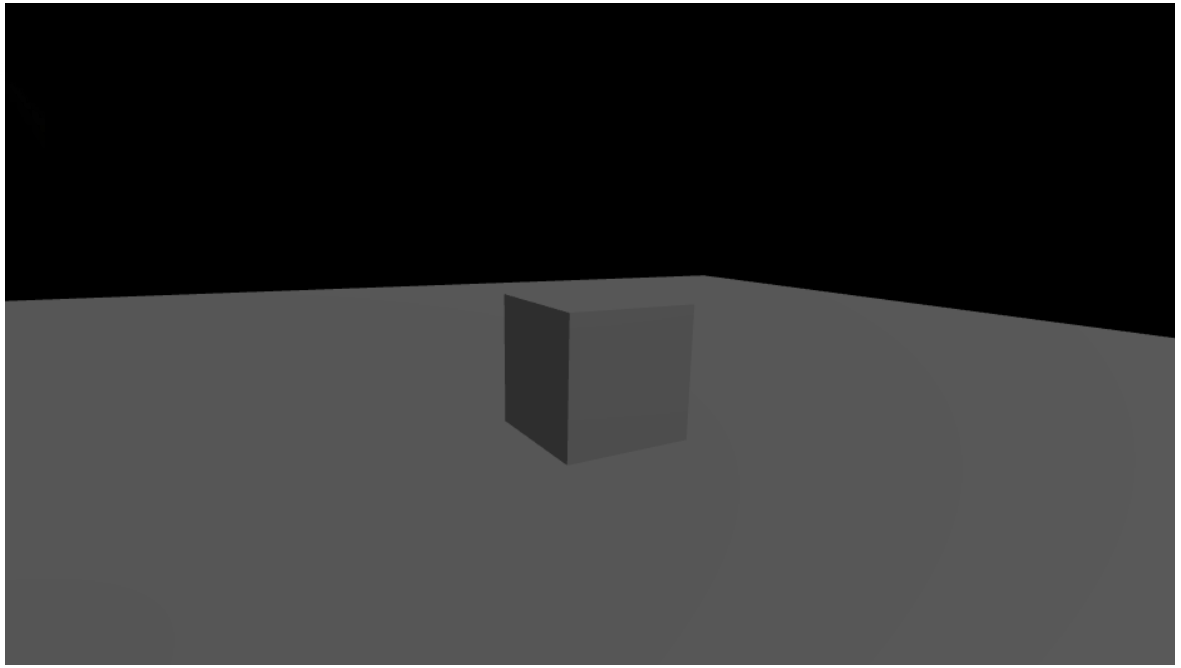
Spot Lights



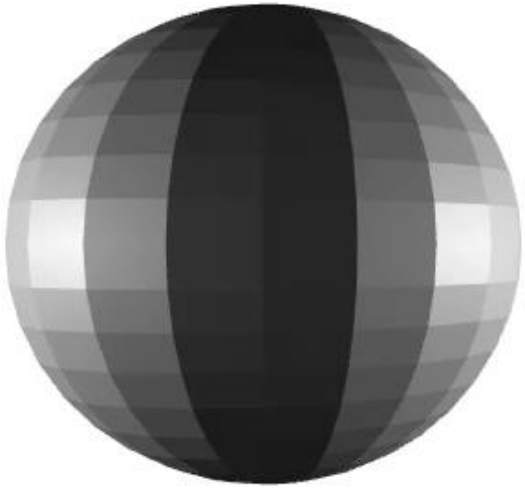


Directional Lights



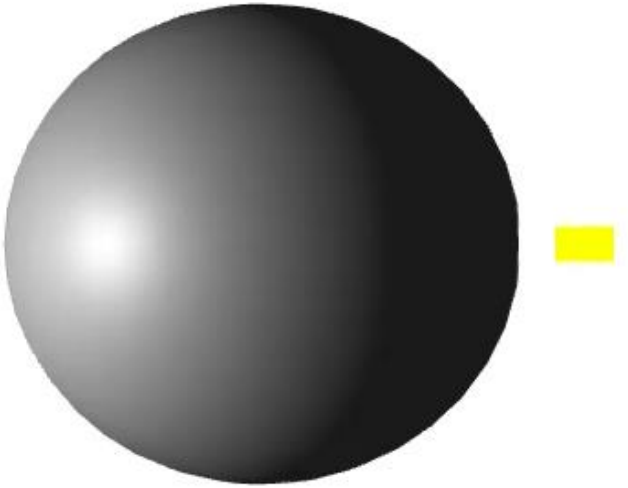


Shading



Flat shading

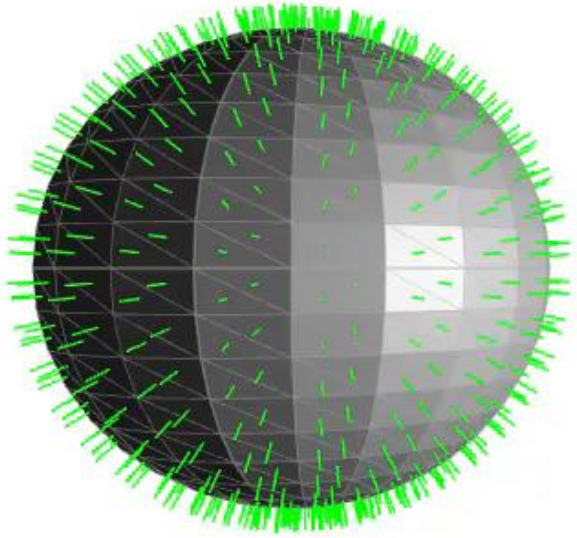
(per vertex)



Smooth shading

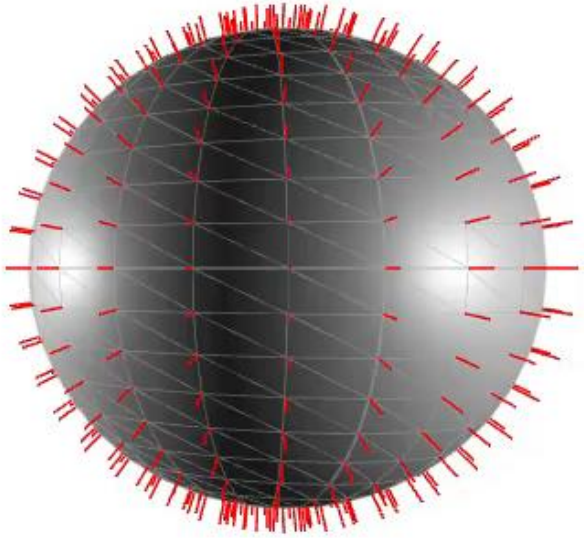
(per pixel)

Shading



Flat shading

(per vertex)



Smooth shading

(per pixel)

Materials

Materials

Materials

Standard
(PBR)

Phong

Lambert

Basic

Basic Material

- Simple material
- Supports coloring
- Transparency
- Alpha maps
- No lighting

Lambert Material

- Everything that is supported in basic material
- Per vertex lighting
- Environment mapping

Phong Material

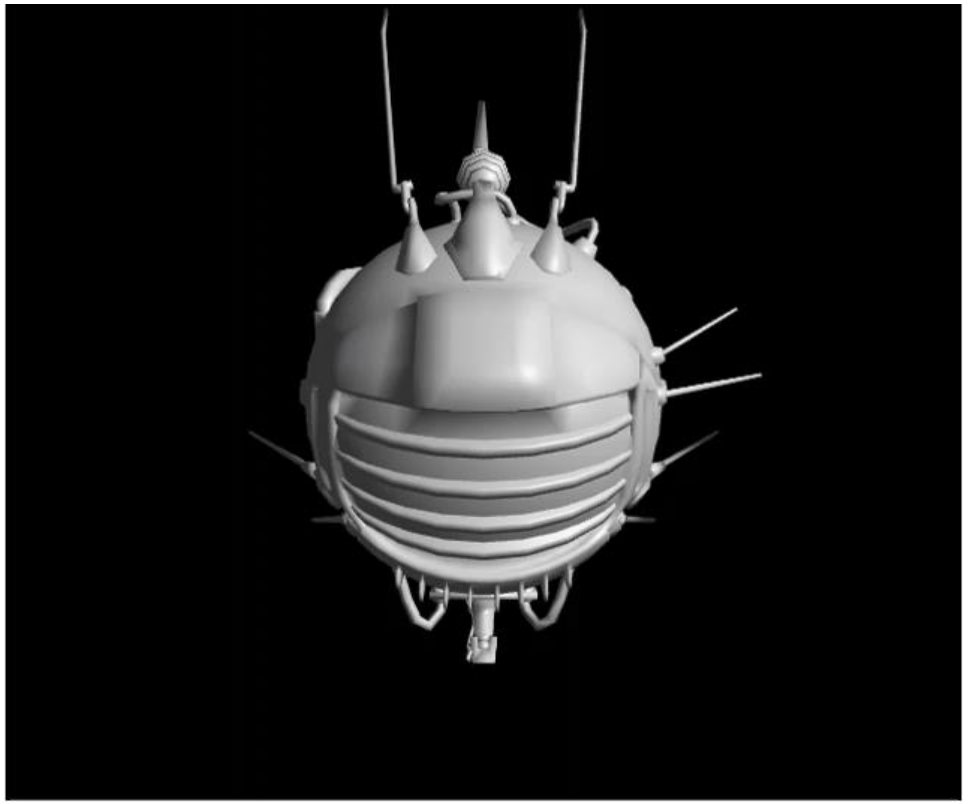
- Per pixel lighting
- Specular component
- Normal maps
- Specular maps
- Displacement maps

PBR Material

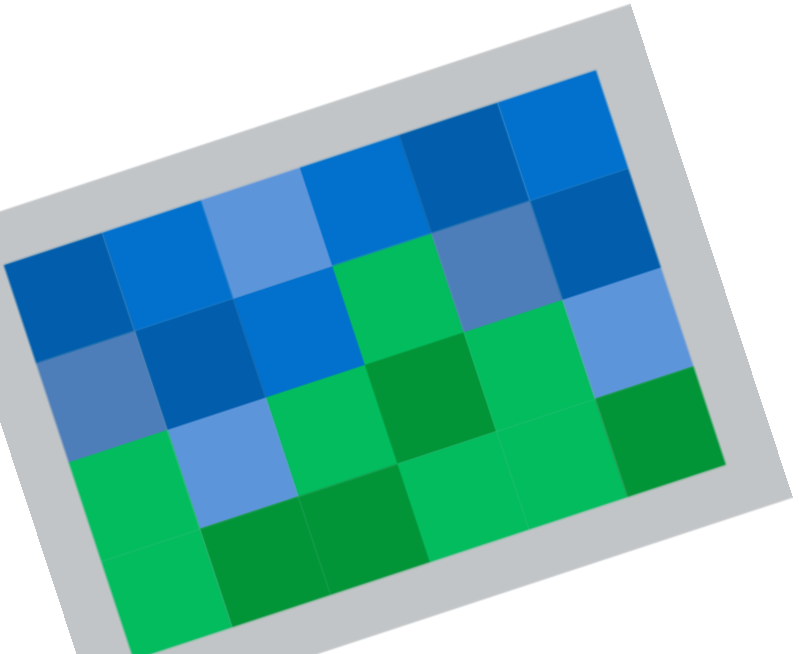
- Metalness and roughness
- Based on a physical model
- Allows designers to collect material characteristics from the real world
- Clear coat

External files

```
var loader = new THREE.OBJLoader();
loader.load("../files/eyebot.obj", function(object)
{
    object.traverse(function(object)
    {
        object.scale.set(0.04, 0.04, 0.04);
        scene.add(object);
    });
});
```



Textures

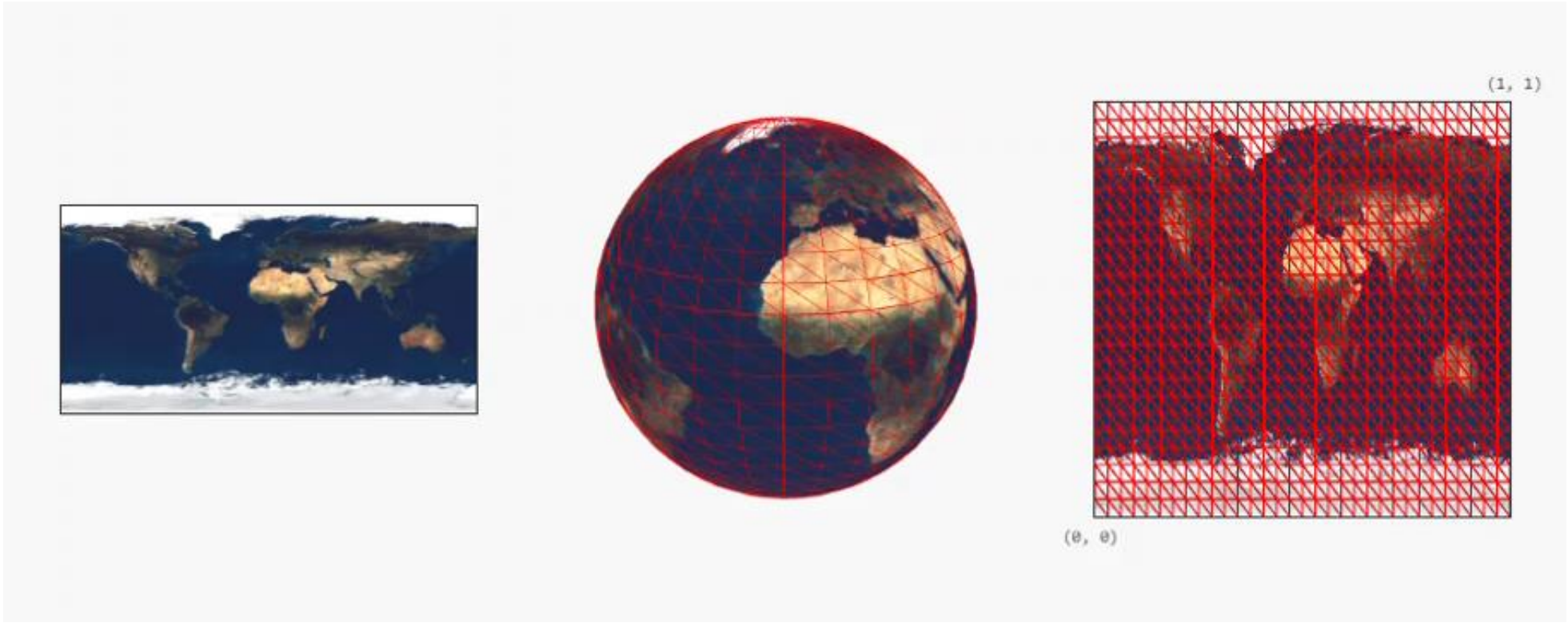


Textures

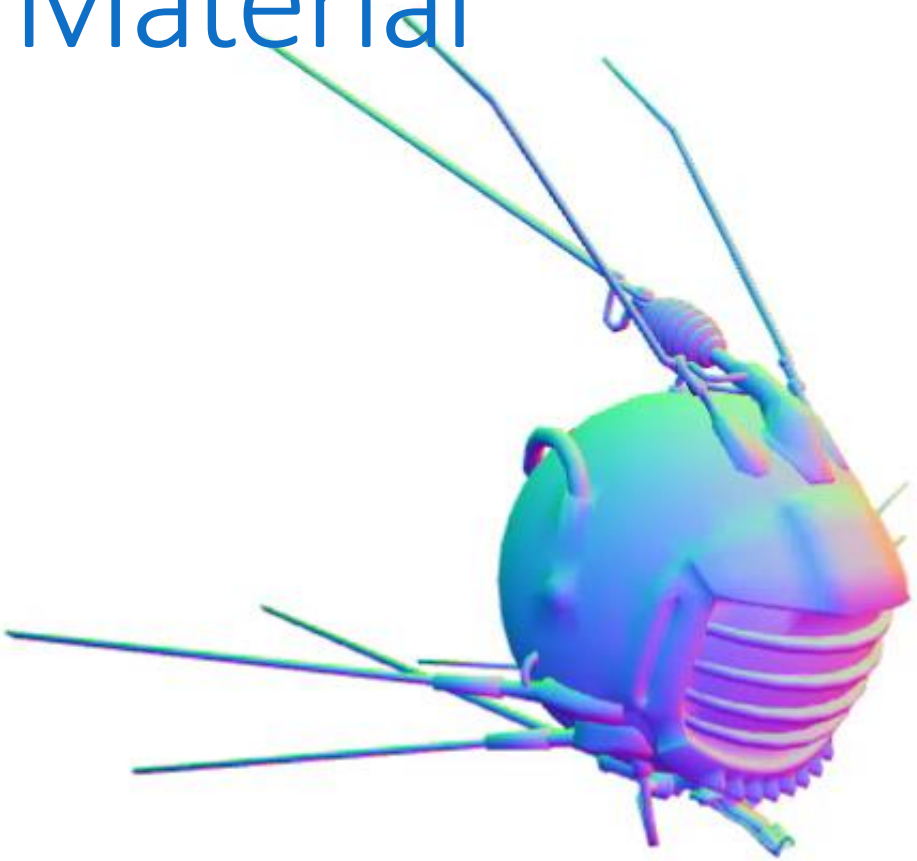
```
//Image
var image = document.createElement("img");
image.src = "../files/texture.png";

//Texture
var texture = new THREE.Texture(image);
image.onload = function()
{
    texture.needsUpdate = true;
}

//Create material
var material = new THREE.MeshPhongMaterial();
material.map = texture;

object.material = material;
```

UV Mapping

UV mapping is the process of projecting a 2D image to a 3D model surface

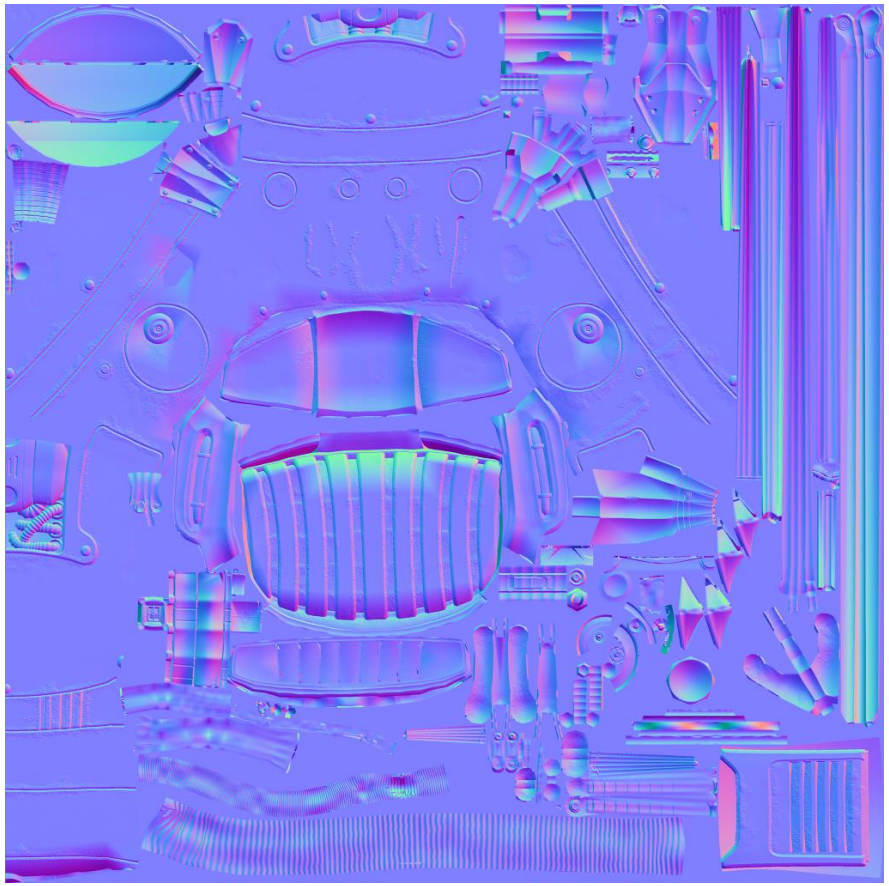


Normal Material



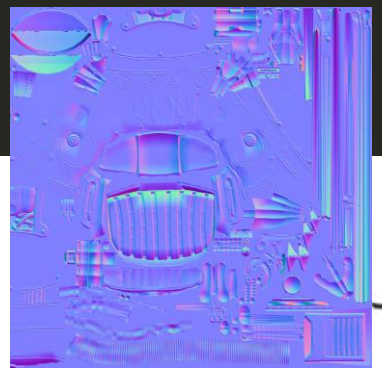
```
new THREE.MeshNormalMaterial()
```

Normal Mapping

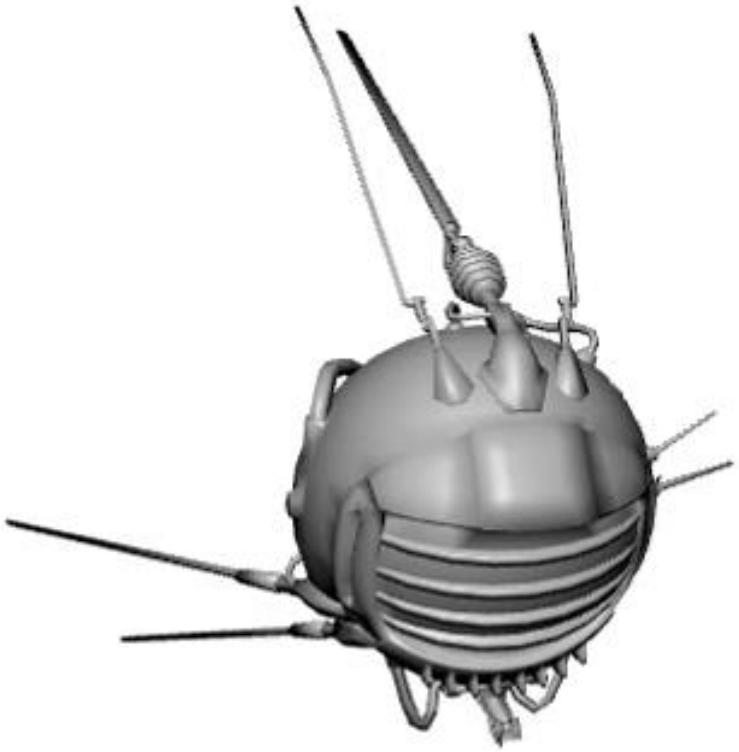


Normal Mapping

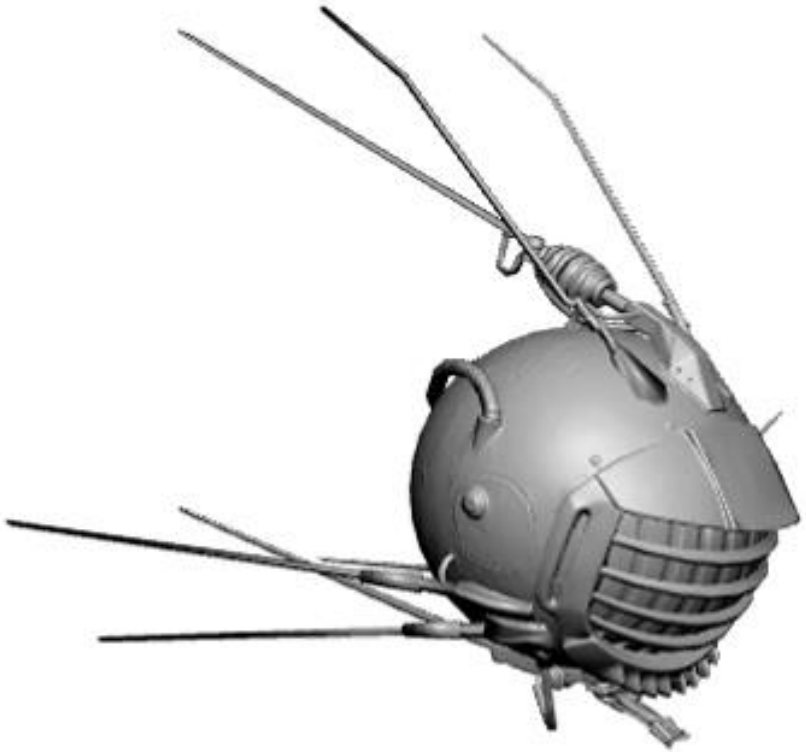
```
var normal = new THREE.Texture(document.createElement("img"));
normal.image.src = "../files/normal.png";
normal.image.onload = function()
{
    normal.needsUpdate = true;
}
material.normalMap = normal;
```



Normal Mapping

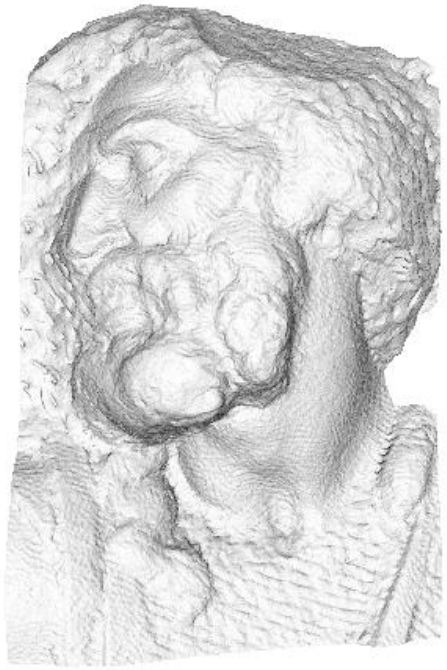


Original



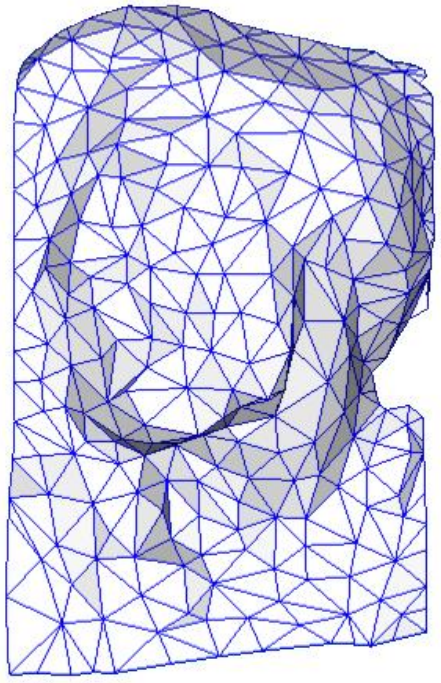
w/ Normal map

Normal Mapping



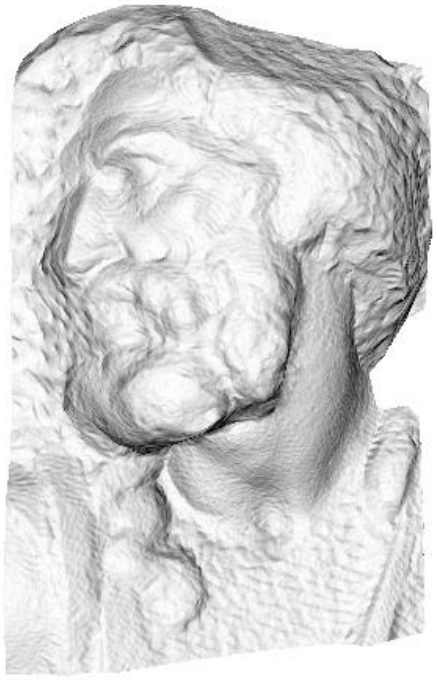
Original

4M Triangles



Simplified

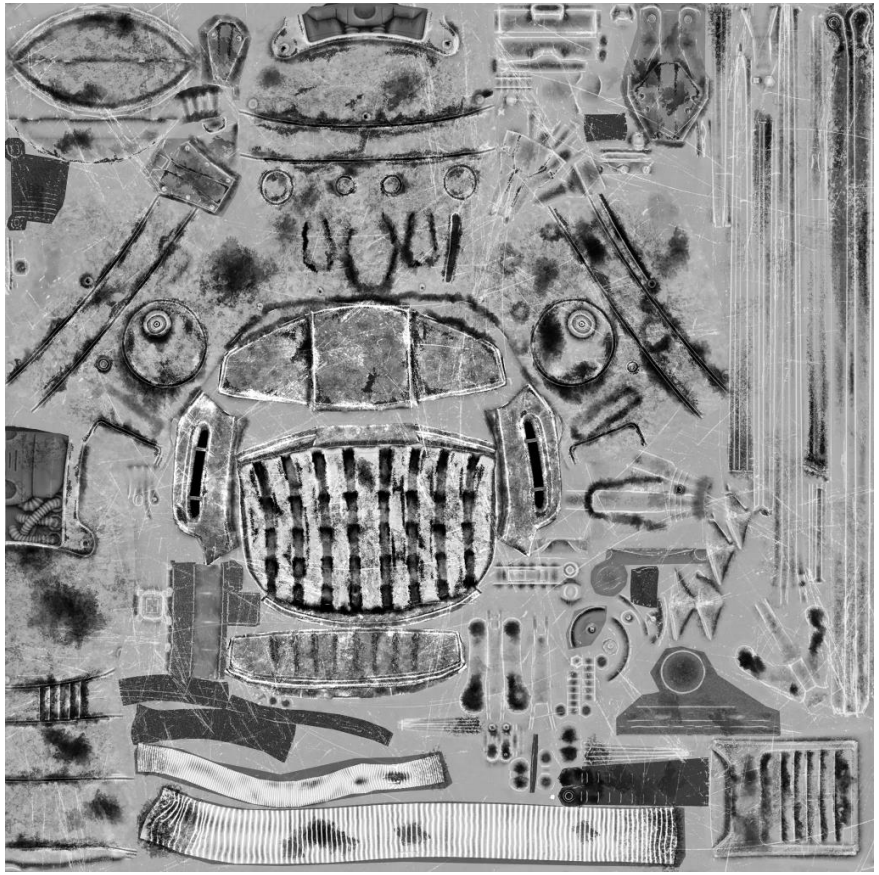
500 Triangles



Simplified

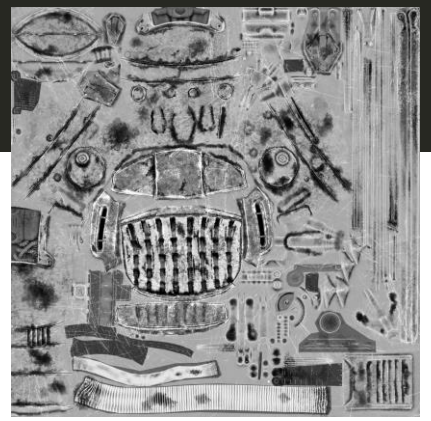
+ Normal map

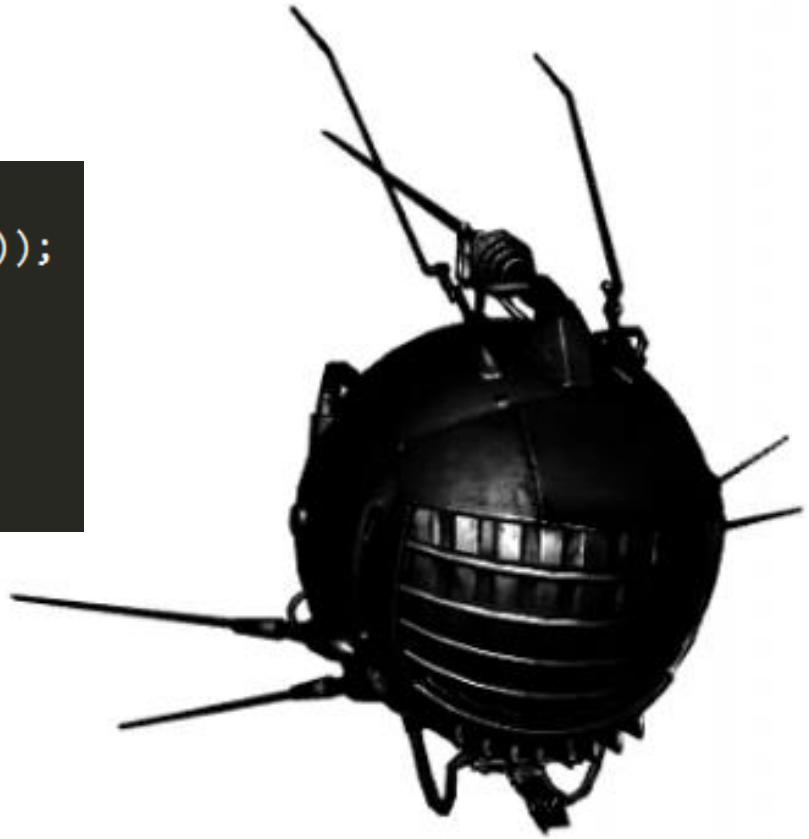
Specular mapping



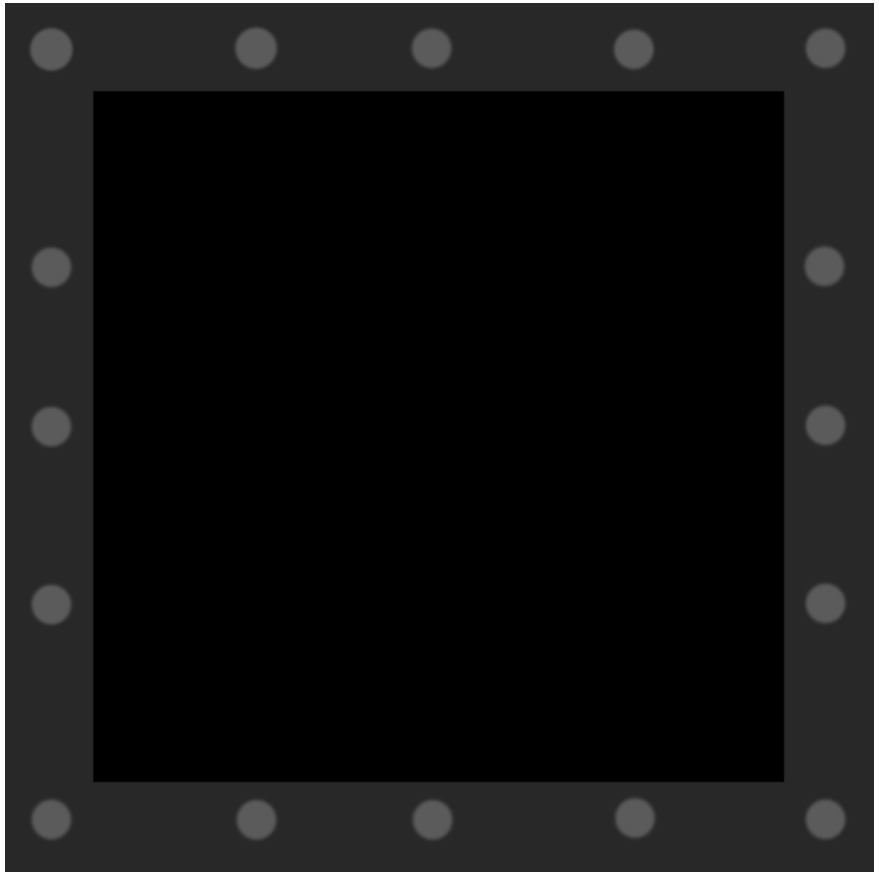
Specular mapping

```
//Specular
var specular = new THREE.Texture(document.createElement("img"));
specular.image.src = "../files/specular.png";
specular.image.onload = function()
{
    specular.needsUpdate = true;
}
```

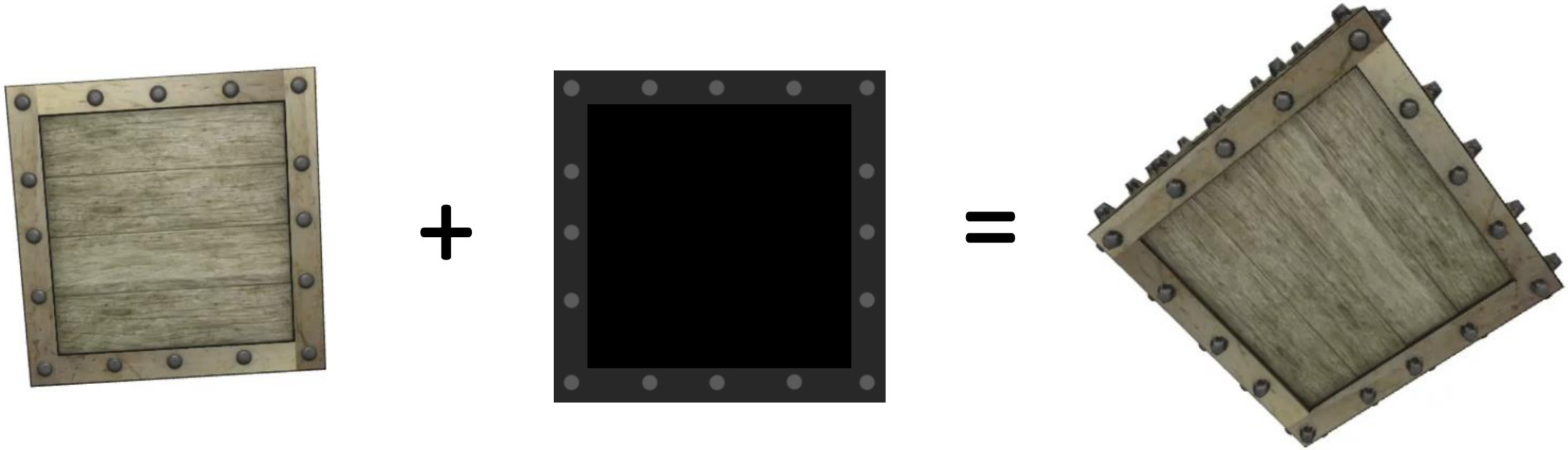




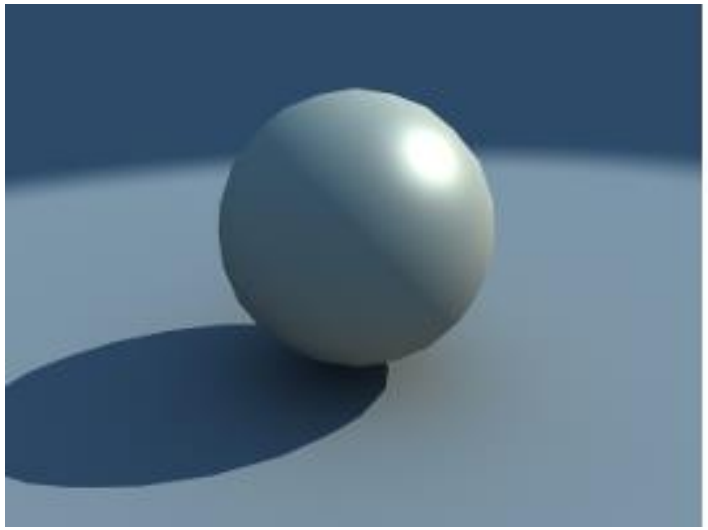
Displacement mapping



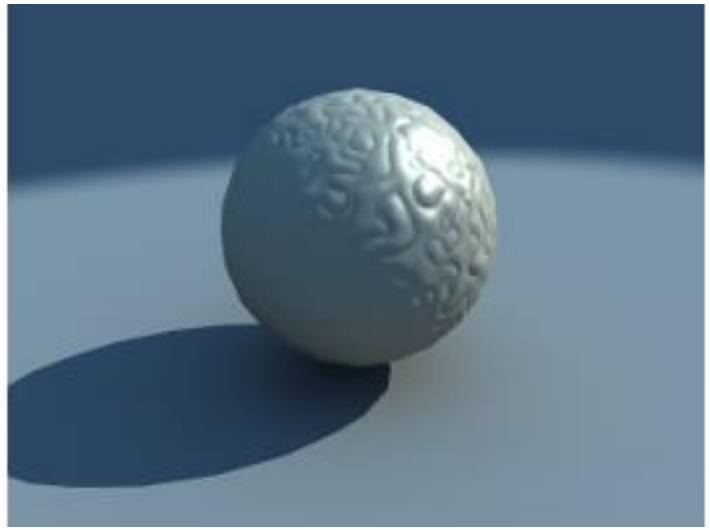
Displacement mapping



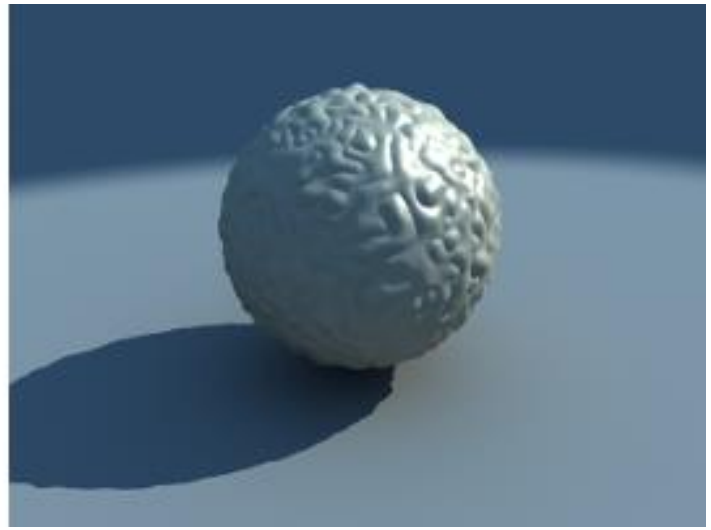
Displacement mapping



Original



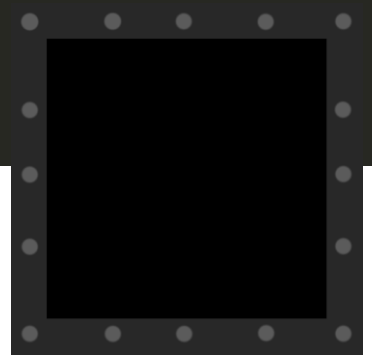
Normal Mapping



Displacement Mapping

Displacement mapping

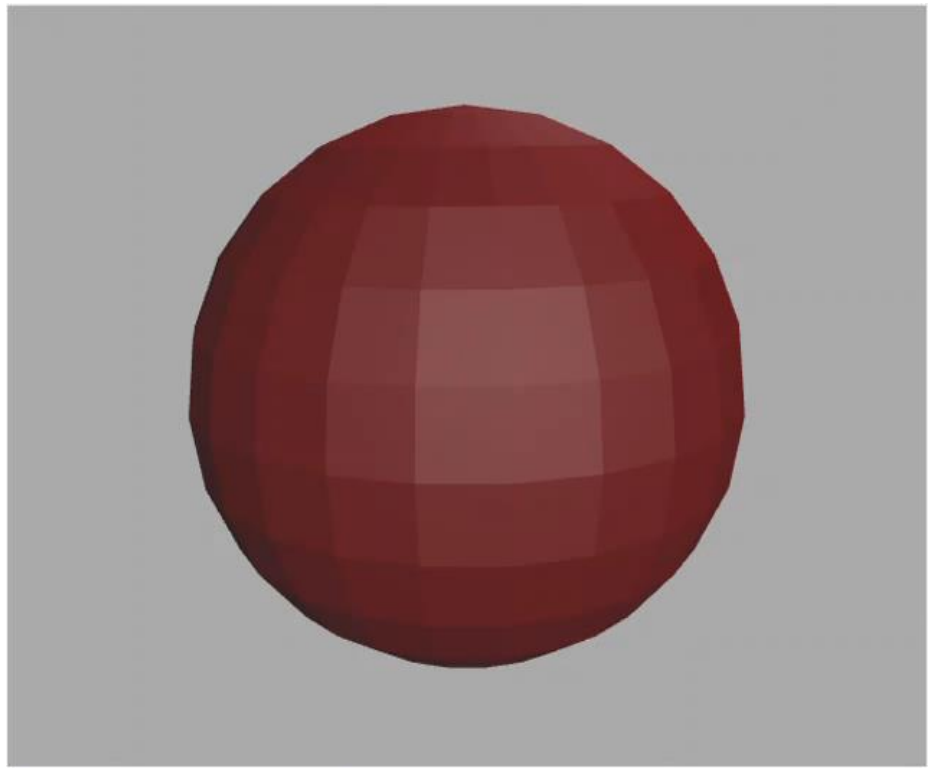
```
var displacement = new THREE.Texture(document.createElement("img"));
displacement.image.src = "../files/crate_displacement.png";
displacement.image.onload = function()
{
    displacement.needsUpdate = true;
}

(...)

material.displacementMap = displacement;
material.displacementScale = 0.2;
material.displacementBias = -0.0315;
```



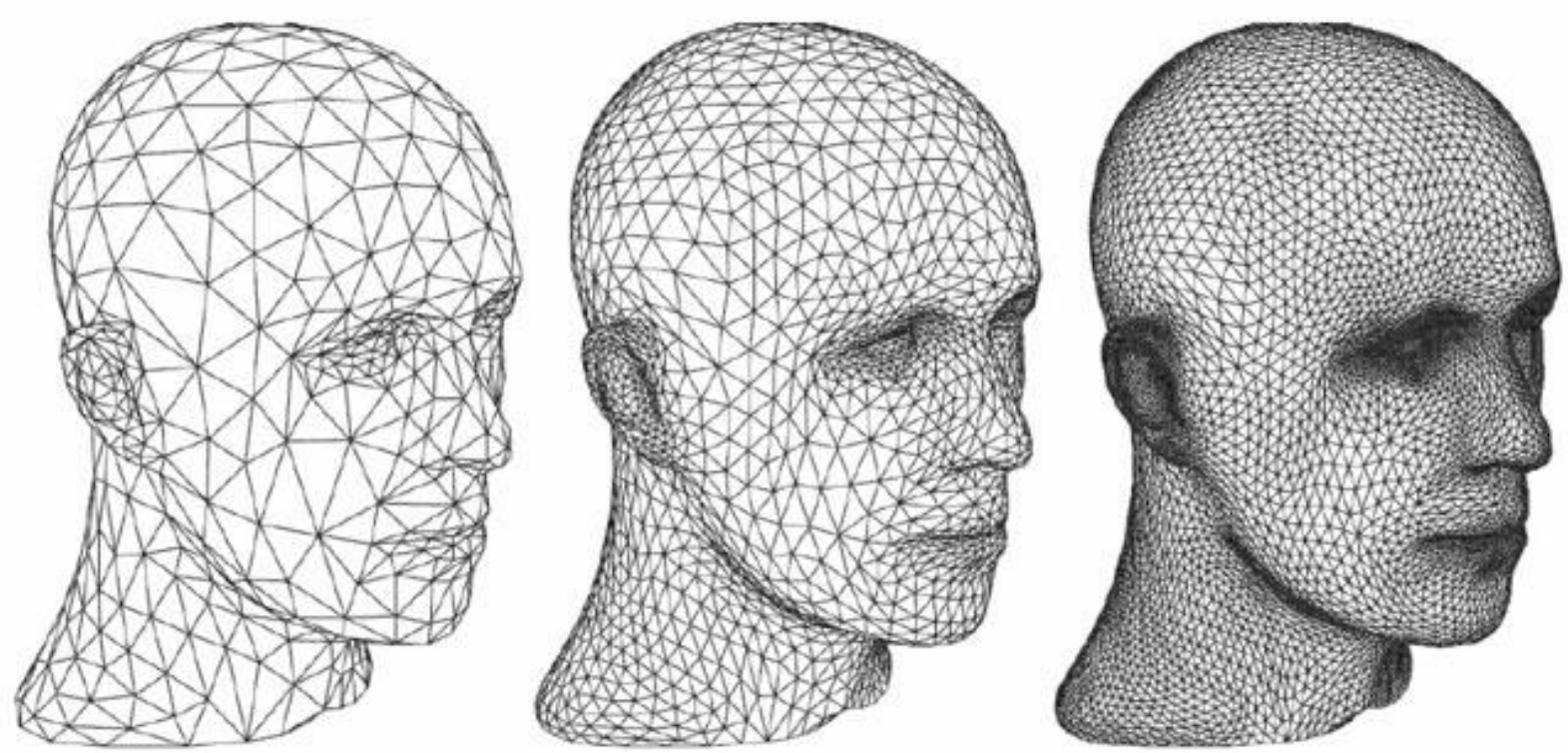
Tesselation

Tesselation

- Consists in repeatedly subdividing the a geometry into a finer mesh
- The GPU generates more vertices and triangles dynamically
- Used with displacement mapping to improve object quality dynamically

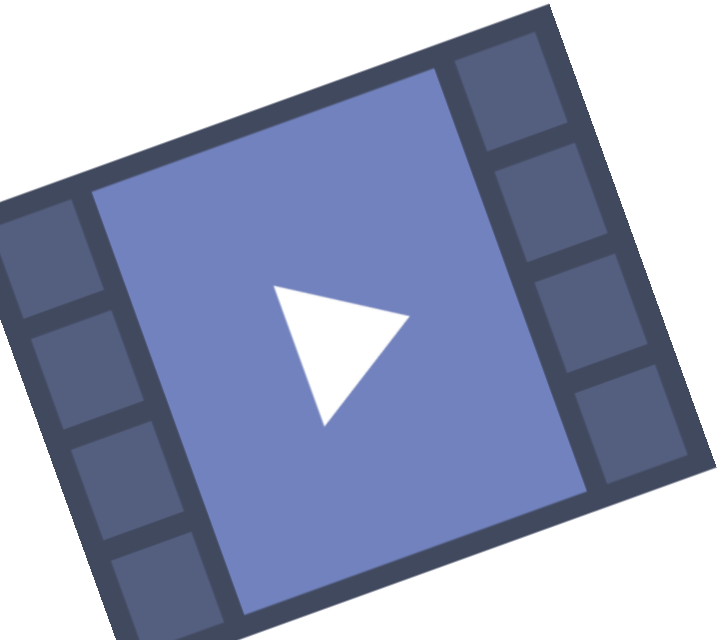


Tesselation



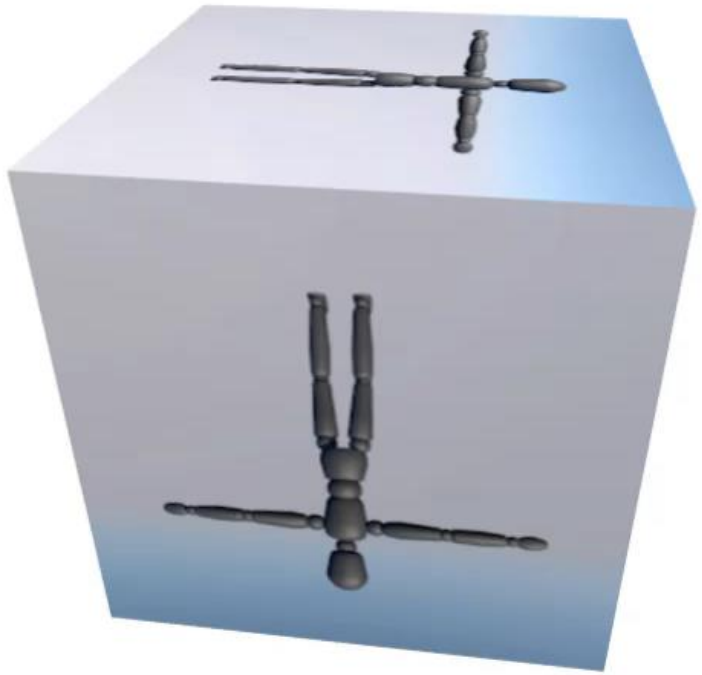
Tesselation

Video Textures



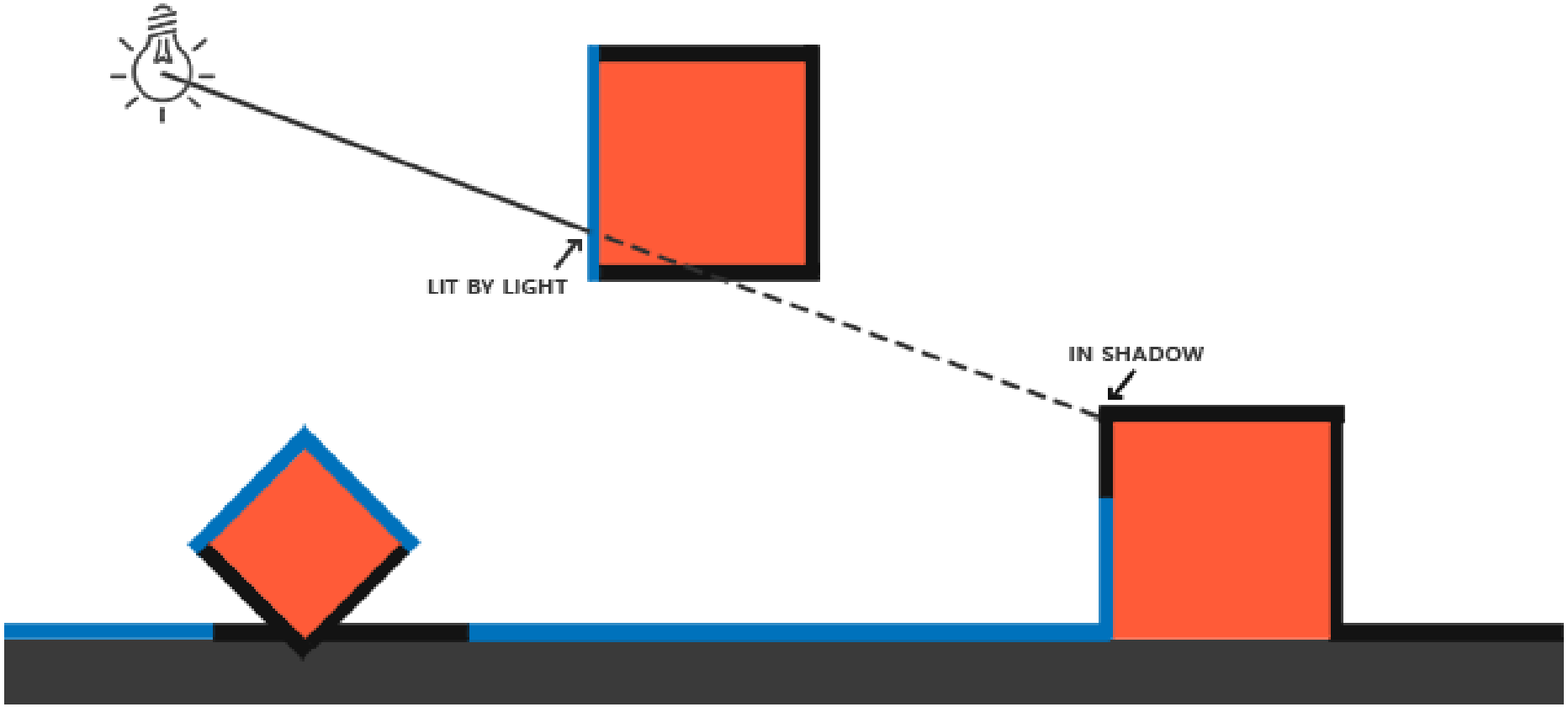
Video Textures

```
var texture = new THREE.VideoTexture(document.createElement("video"));
texture.minFilter = THREE.LinearFilter;
texture.image.src = "../files/video.webm";
texture.image.loop = true;
texture.image.autoplay = true;
texture.image.onload = function()
{
    texture.needsUpdate = true;
}
```

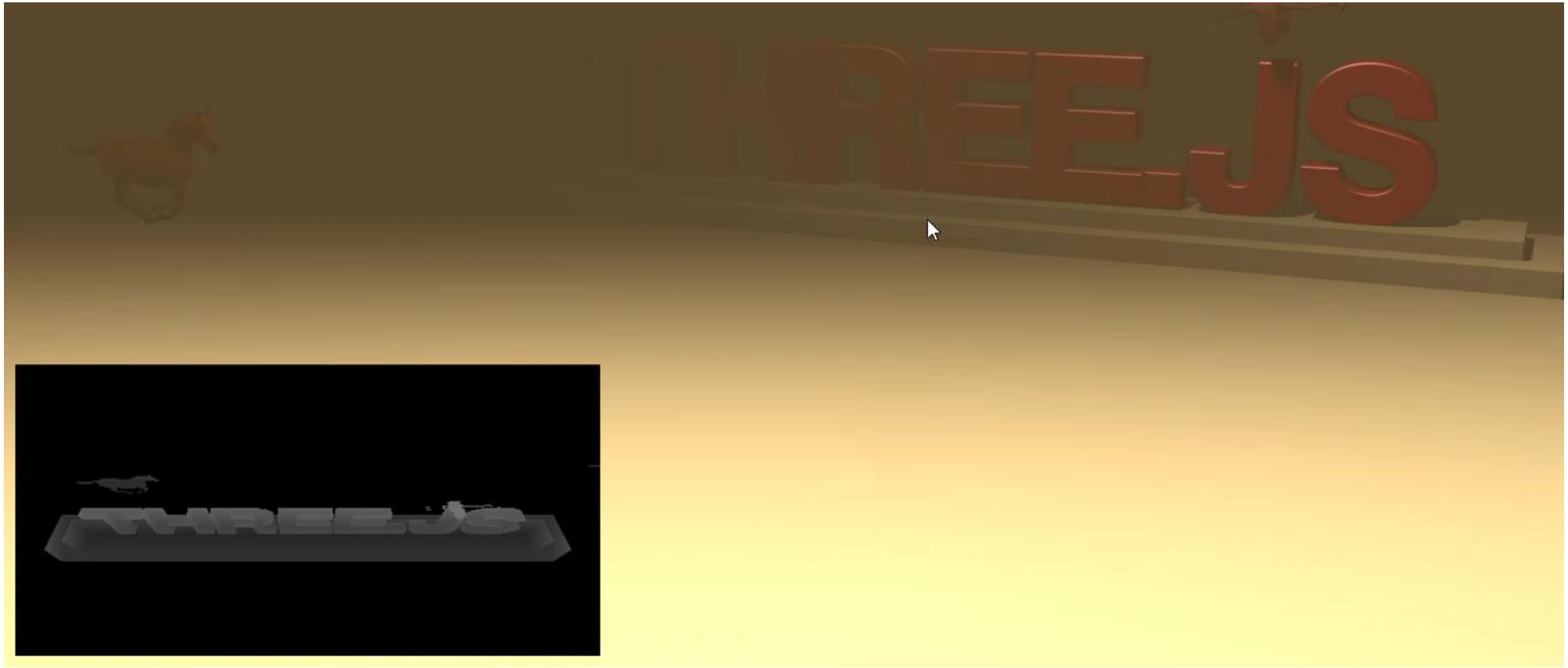


Shadows

Shadows



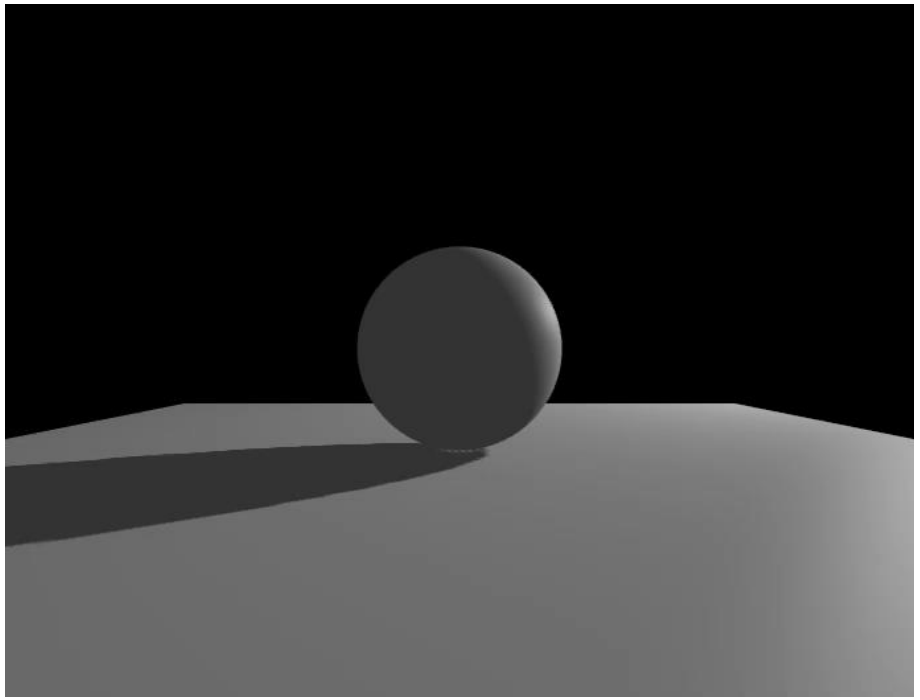
Shadow map (threejs.org/examples/webgl_shadowmap.html)



Shadows

Point, directional and spot ligths can cast shadows

```
renderer.shadowMap.enabled = true;
renderer.shadowMap.type = THREE.PCFSoftShadowMap;
(...)
sphere.castShadows = true;
ground.castShadows = true;
ground.receiveShadows = true;
(...)
ligth.castShadows = true;
```



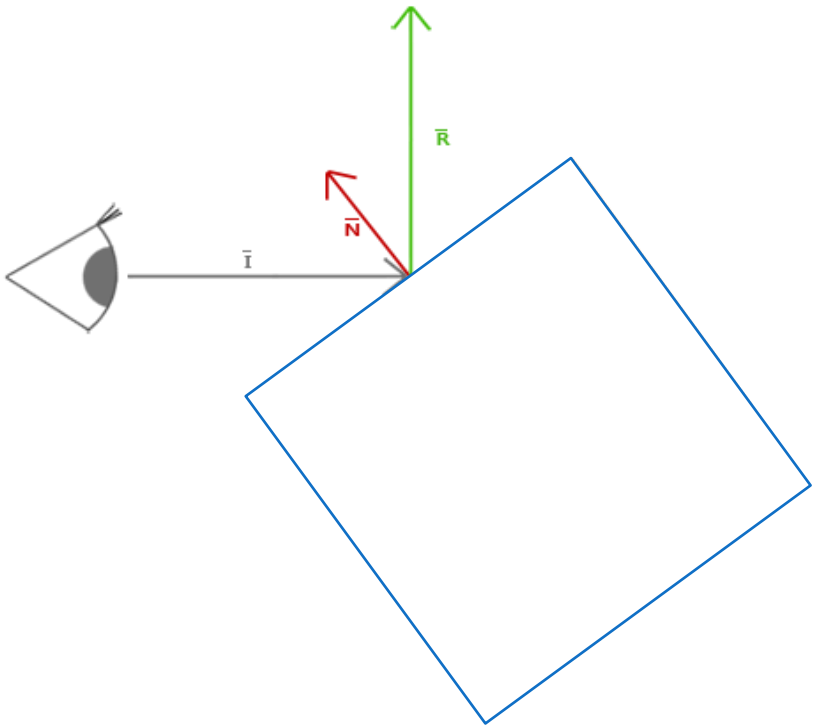
Cube mapping

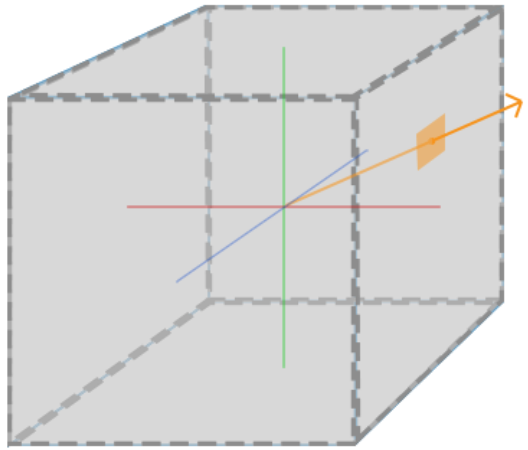
Cube mapping

Cube mapping

```
//Cube map
var path = "../files/cube/";
var format = ".jpg";
var urls =
[
    path + "px" + format, path + "nx" + format,
    path + "py" + format, path + "ny" + format,
    path + "pz" + format, path + "nz" + format
];

var cube = new THREE.CubeTextureLoader().load(urls);
cube.format = THREE.RGBFormat;

scene.background = cube;
```

Reflections

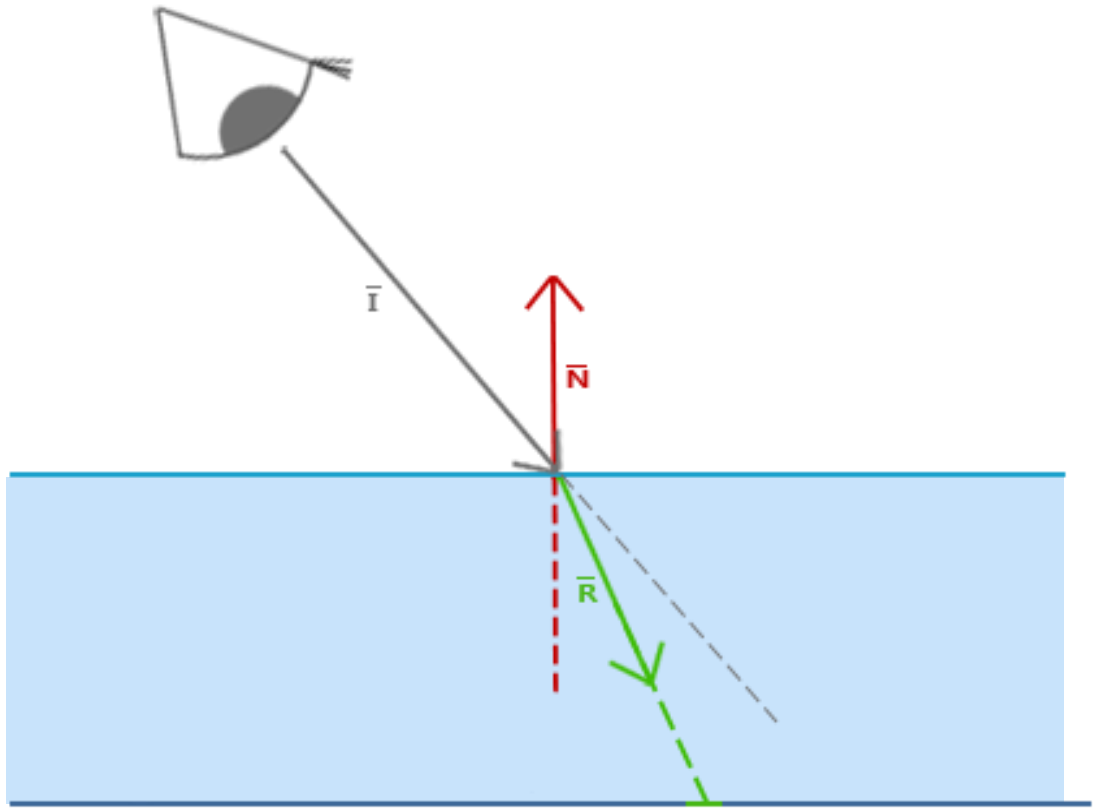




Reflections

```
(...)

material.normalMap = normal;
material.envMap = cube;
material.combine = THREE.MixOperation;
material.reflectivity = 1.0;

(...)
```

Refractions



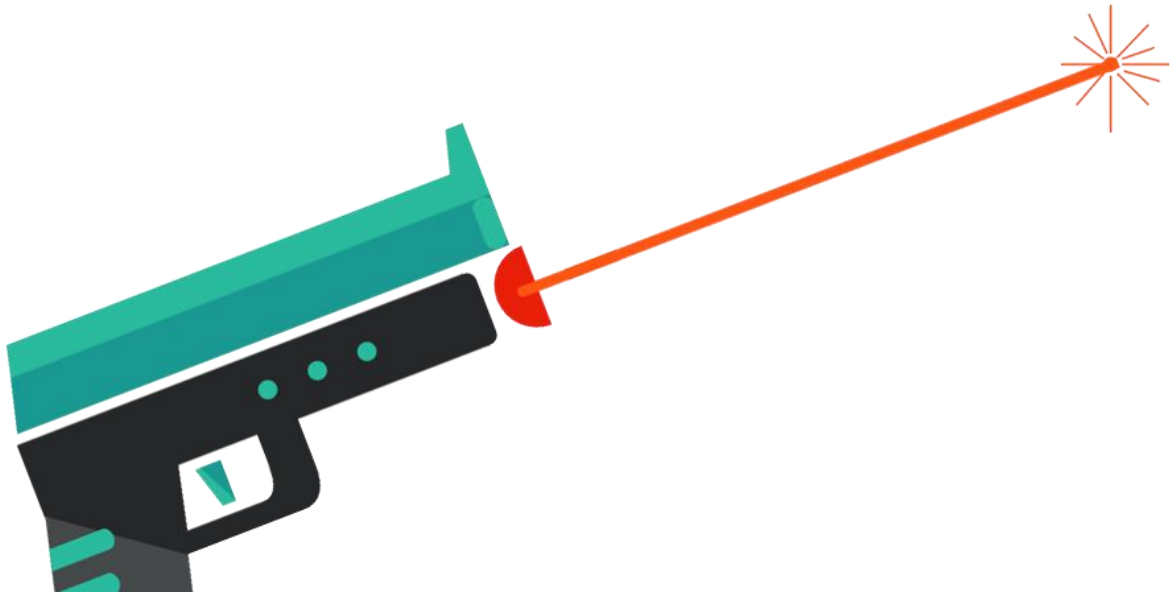
Refractions

```
cube.mapping = THREE.CubeRefractionMapping;

(...)

material.combine = THREE.MixOperation;
material.refractionRatio = 0.9;
```

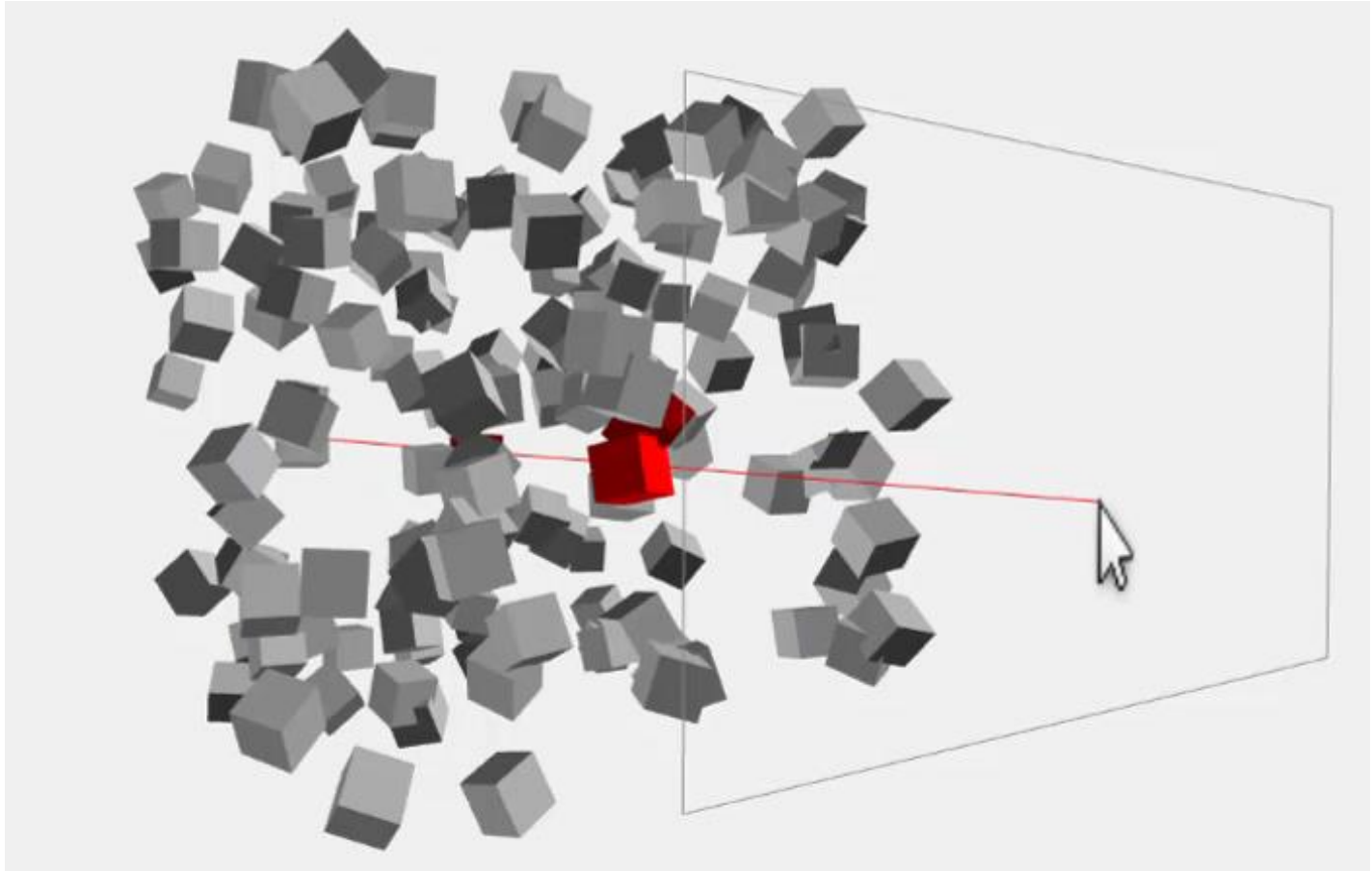
Cube cameras

```
//Cube camera
var cube_camera = new THREE.CubeCamera(0.1, 1000, 256);
(...)
material.envMap = cube_camera.renderTarget.texture;
(...)
eyebot.visible = false;
cube_camera.position.copy(eyebot.position);
cube_camera.updateCubeMap(renderer, scene);
eyebot.visible = true;
```

Raycaster

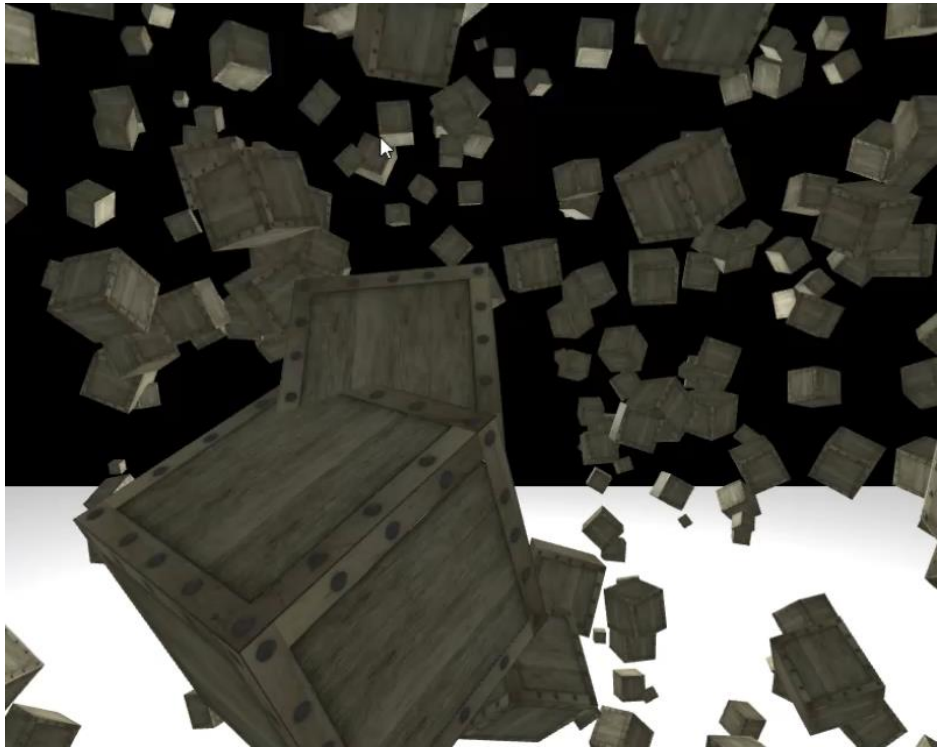


Raycaster



Raycaster

```
var raycaster = new THREE.Raycaster();
var normalized = new THREE.Vector2();

(...)

//Normalize mouse coordinates (range -1 to 1)
normalized.x = 2 * (mouse.position.x / canvas.width) - 1;
normalized.y = 1 - 2 * (mouse.position.y / canvas.height);

//Update raycaster ray position relative to camera
raycaster.setFromCamera(normalized, camera);

//Get intersected objects
var intersects = raycaster.intersectObjects(scene.children);
for(var i = 0; i < intersects.length; i++)
{
    intersects[i].object.material.color.set(0xFFAAAA);
}
```

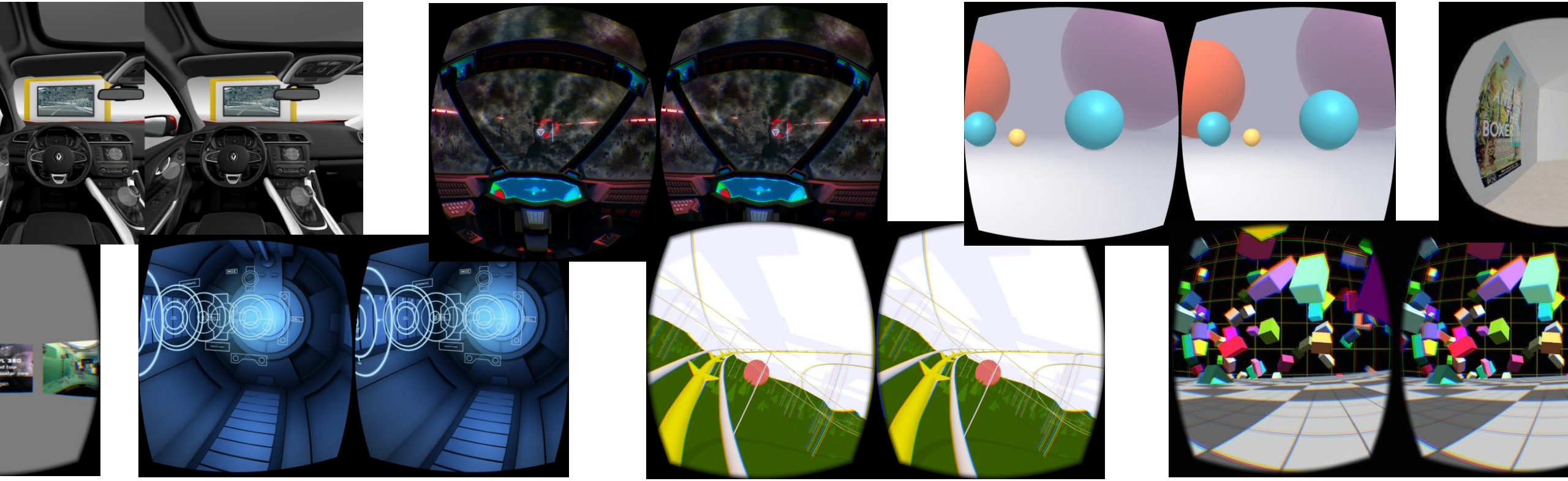


WebVR

WebVR (w3c.github.io/webvr/)

- WebVR is an JavaScript API that provides access to Virtual Reality devices, in your browser.
- WebVR is still an experimental feature
  - Special builds of browsers are required for now

WebVR



WebVR

Microsoft Edge

(Announced)

Google Chrome

WebVR 1.1

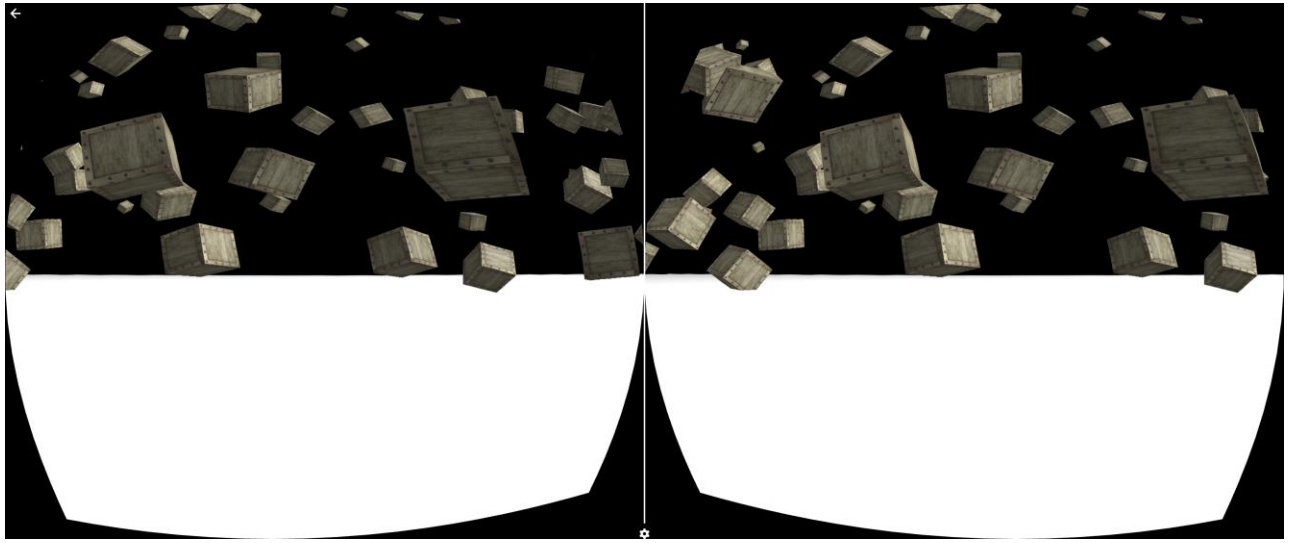
Mozilla Firefox

WebVR 1.1

Samsung Gear VR Browser

WebVR

```
var controls = new THREE.VRControls(camera);
var effect = new THREE.VREffect(renderer);

(...)

//Update camera rotation and pos from HMD
controls.update();

//Render scene to HMD
effect.render(scene, camera);
```



WebVR

Performance tips

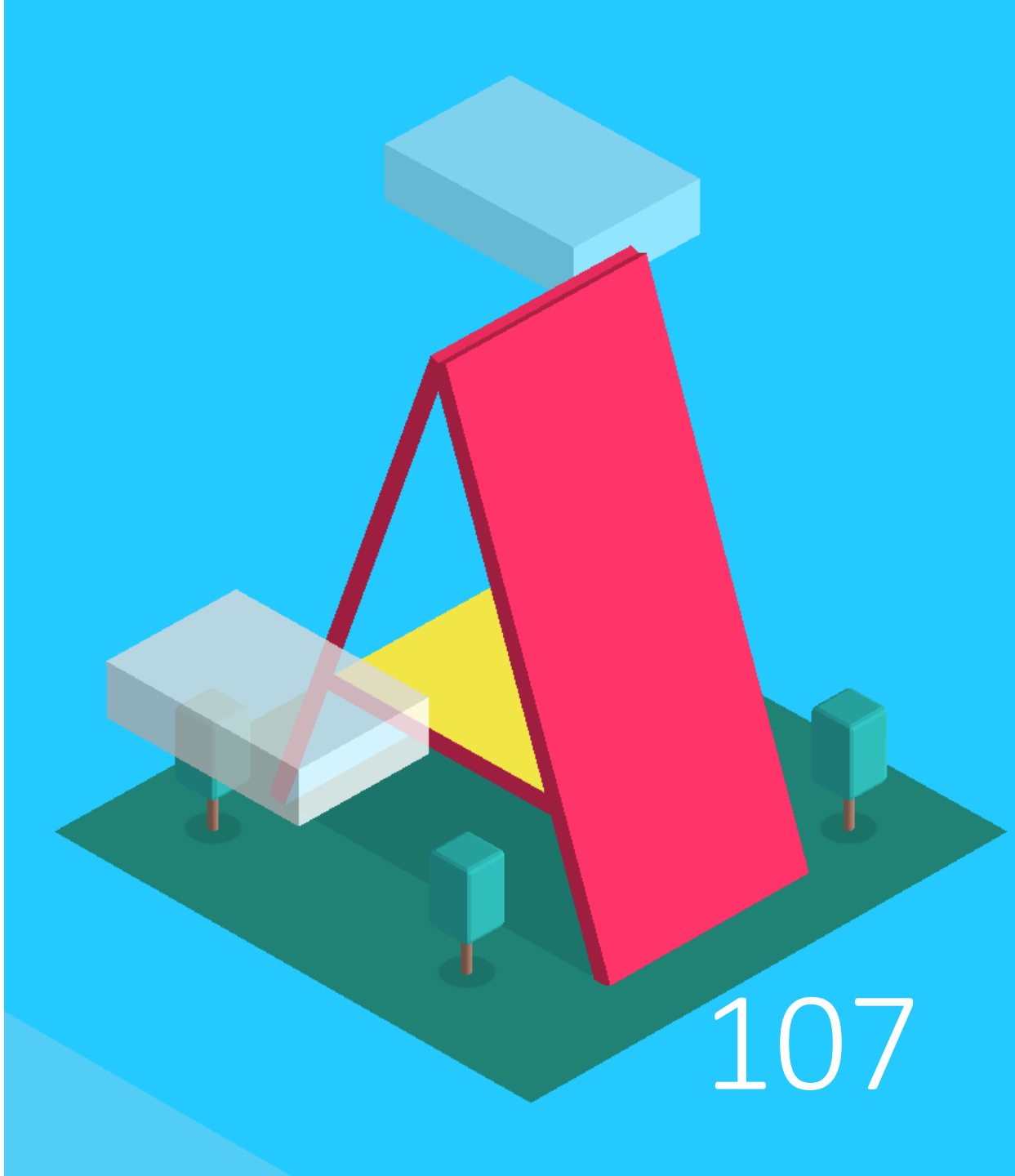
- Polygon count
  - Reduce polygon count and use normal maps
- Buffered Geometry vs Geometry
- Material performance
  - Basic and Lambert materials peform better
- Object count
  - Merge geometries when possible

Whats next?

A-Frame

- VR Web Apps written in HTML
- Entity-component ecosystem
- Built on top of three.js

```
<!--Objects-->
<a-sphere position="0 1.25 -1" radius="1.25" color="#EF2D5E"></a-sphere>
<a-cylinder position="1 0.75 1" height="1.5" color="#FFC65D"></a-cylinder>
<a-plane rotation="-90 0 0" width="4" height="4" color="#7BC8A4"></a-plane>

<!--Sky and camera-->
<a-sky color="#ECECEC"></a-sky>
<a-entity position="0 0 3.8">
    <a-camera></a-camera>
</a-entity>
```



WebGL 2.0

- Based on OpenGL 3 ES

Questions?

References

THREE.JS - http://threejs.org/ (September 18, 2016)

AlteredQualia - http://alteredqualia.com/ (September 18, 2016)

Open.gl - https://open.gl/drawing (October 04, 2016)

OpenGL Tutorial - http://www.opengl-tutorial.org/ (October 04, 2016)

FlatIcon - http://www.flaticon.com/ (October 04, 2016)

nunuStudio - https://github.com/tentone/nunustudio (October 05, 2016)

Acko.net - https://acko.net/ (October 05, 2016)

Oculus - https://developer.oculus.com/webvr/ (October 05, 2016)

NVIDIA - http://www.nvidia.com/object/tessellation.html (October 05 2016)