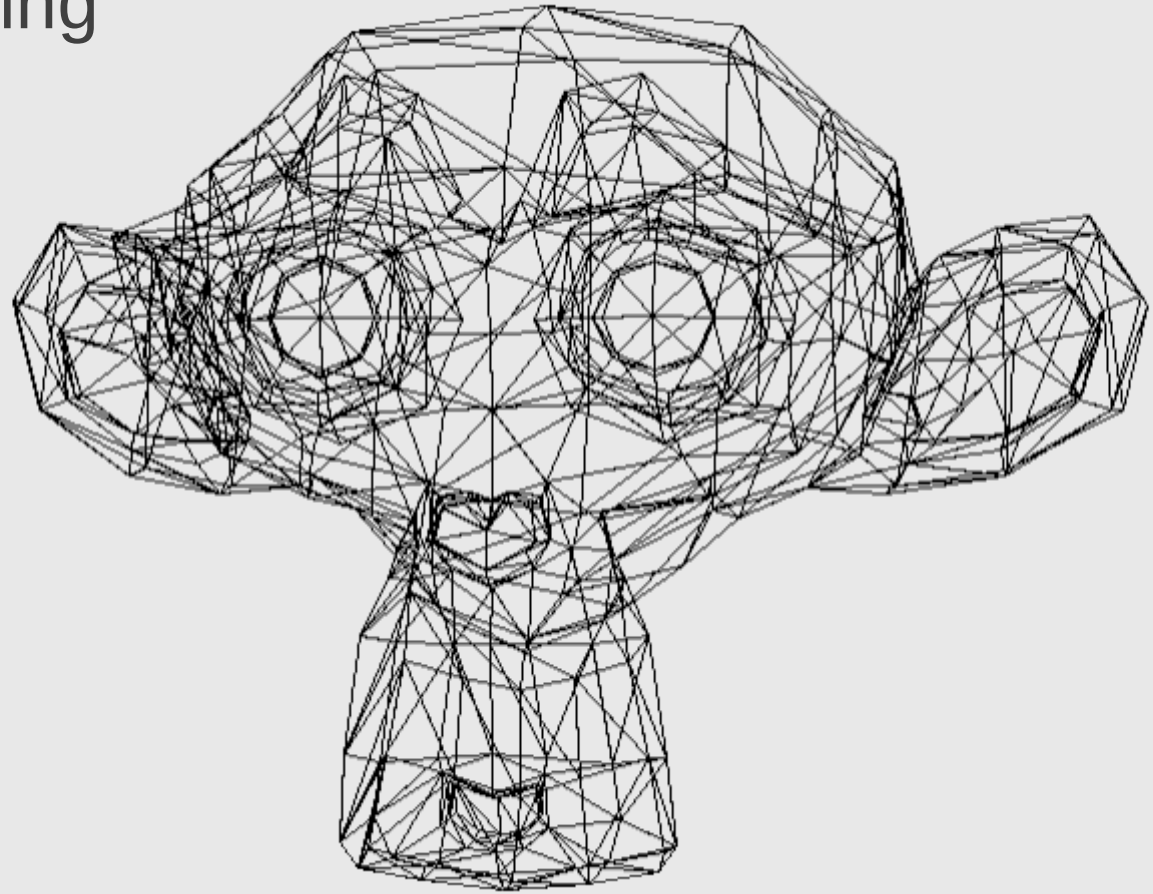
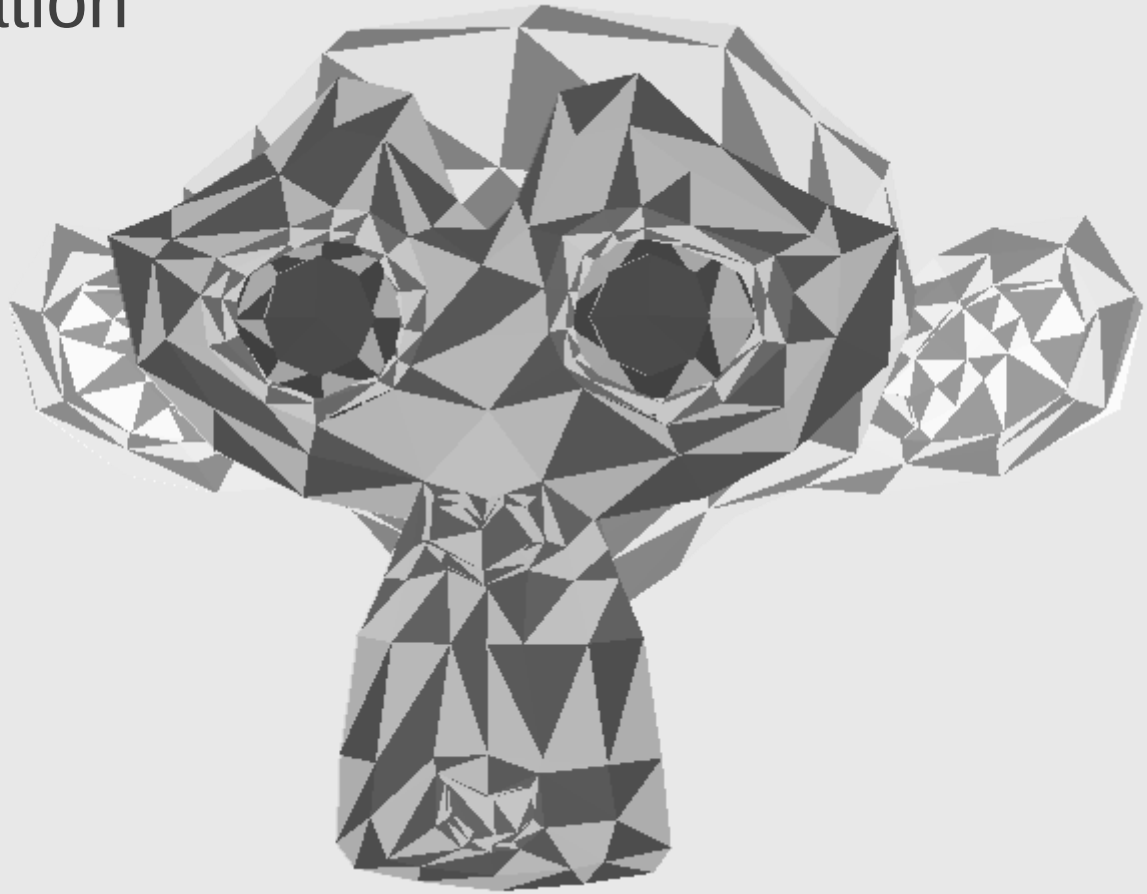
3D on the Web : Understanding the basics

Introduction to WebGL 3D with HTML5 and Babylon.js

Wireframing

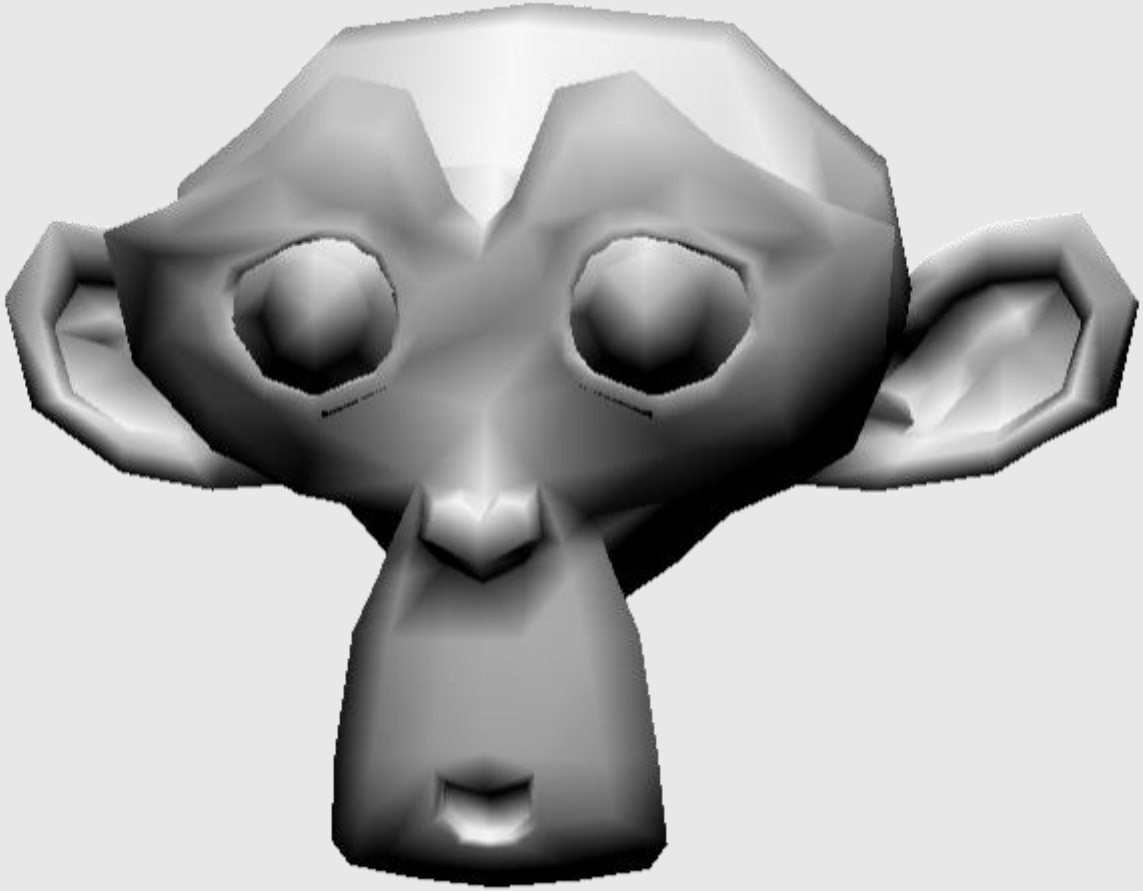


Rasterization



Flat Shading

Gouraud Shading



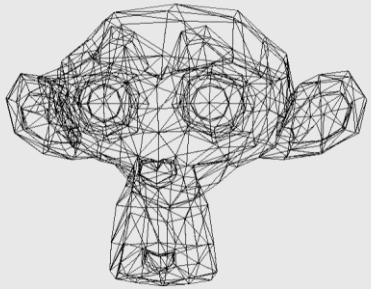
Texture mapping

Step One

Understanding the transformation pipeline

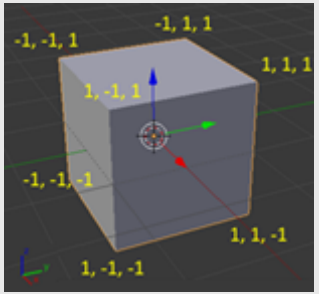
Some 3D engine vocabulary

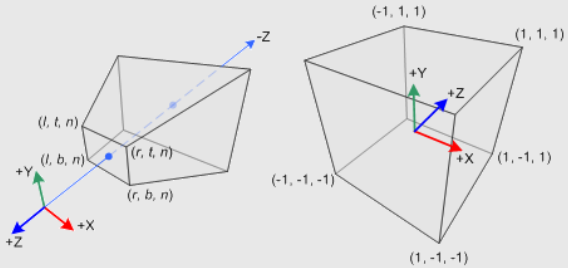
- A point in the 3D world = a **vertex**
- Multiple vertex = **vertices**
- **Vector3** (x,y,z) is used for a 3D position or a direction
- Triangle = **face**
- A 3D object = a **mesh**



Spaces

- Euclidean space using **Cartesian coordinates** : X, Y and Z
- **Local**/Model Space
- **World** Space
- View/**Camera** Space (Point-of-view)
- Screen space (**2D**)



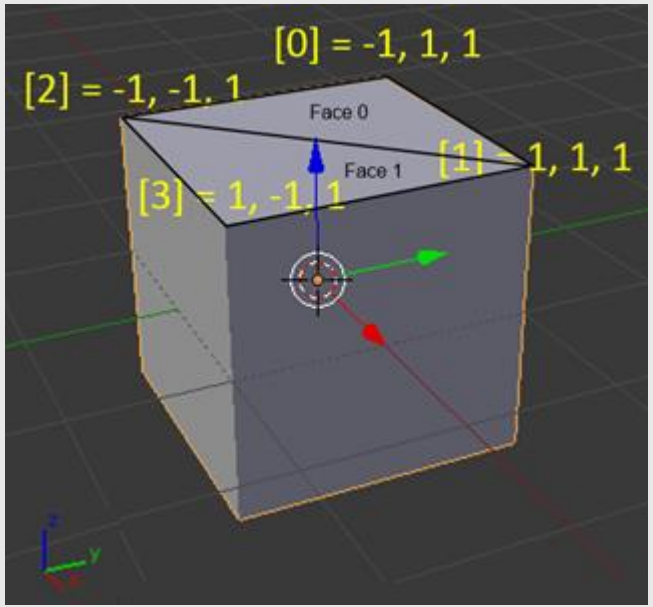


Step Two

It's all about triangles

Drawing triangles for a cube

- A **cube** is made of **8 vertices**
- Each **face** is made of **3 vertices**
- A **cube** is made of **12 faces**

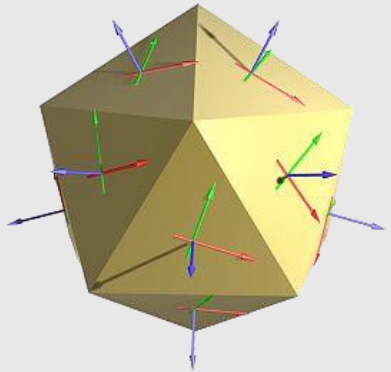


Step Three

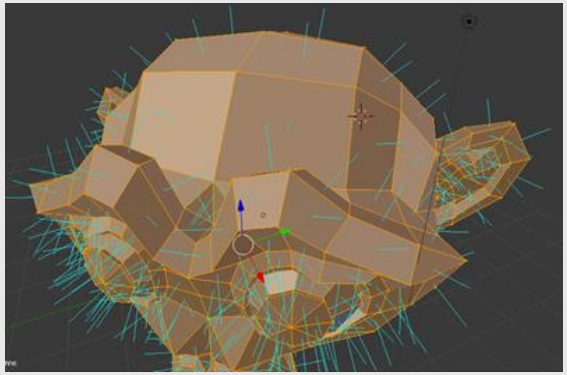
Filling the triangles with the proper pixels

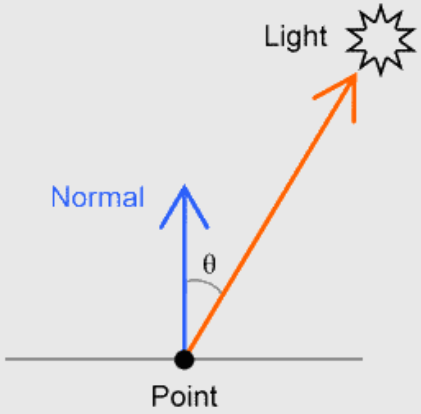
Rasterization

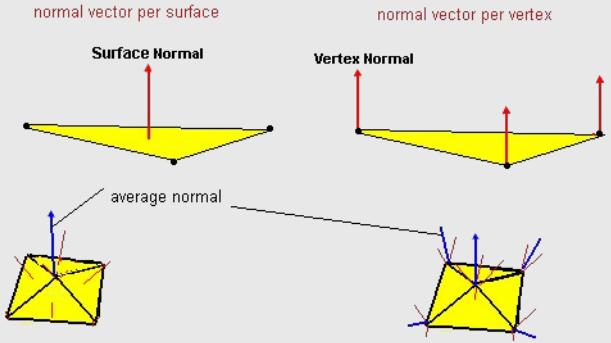
- Algorithm to **fill the triangle** with lines
- Pay attention to **Z-order** via a depth buffer
- To add lights & shadows we need **normals**
- There are different kinds of shading algorithms like:
  - **Flat** Shading: 1 normal per face, on its center
  - **Gouraud** Shading: 3 normals per face on each vertex, using interpolation to compute the color via gradients

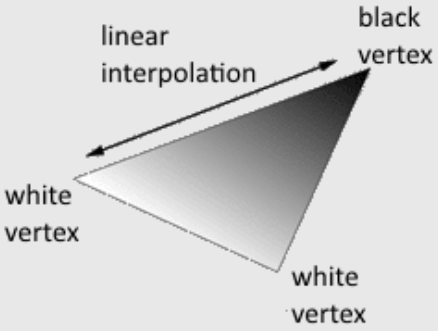


Flat & Gouraud Shading explained

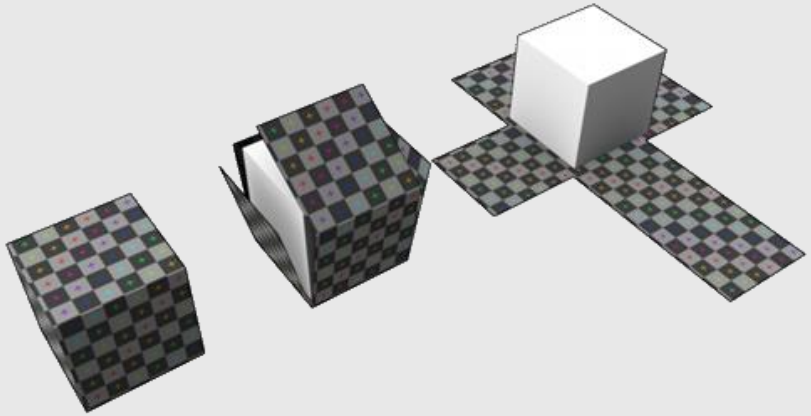




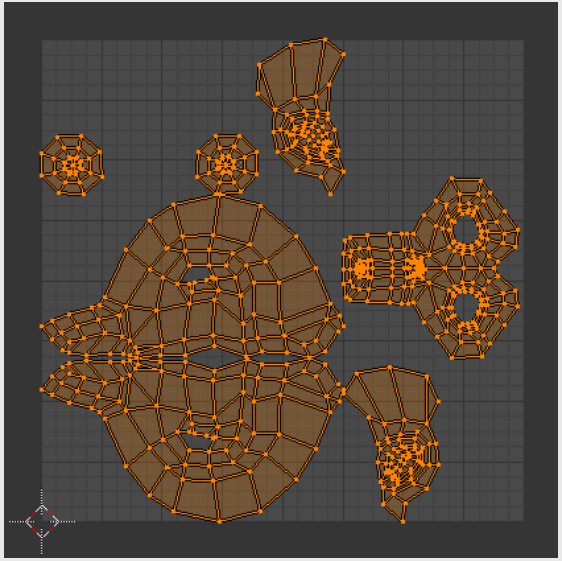


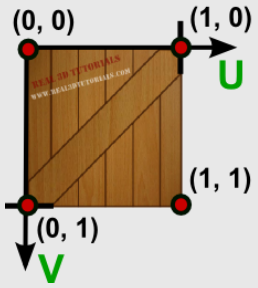


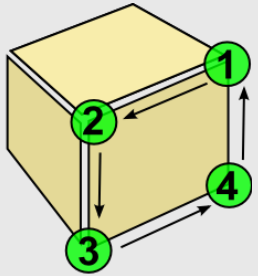
Texture mapping – basic concepts



Texture mapping – unwrapping & UV mapping







Section Two

Moving from CPU to GPU

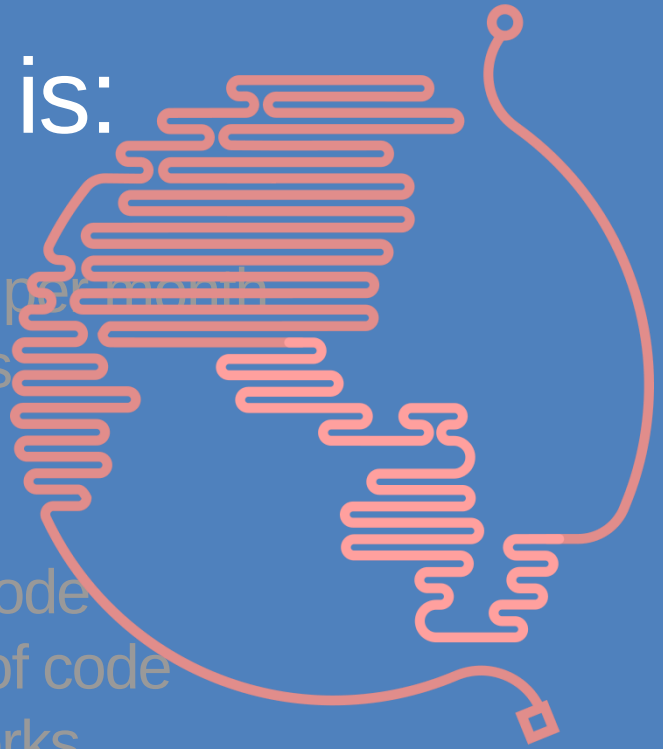
The rise of GPUs

Hardware accelerated rendering:
2D Canvas, CSS3 animations

H264 & JPG hardware decoding

Accelerated 3D
with WebGL

Babylon.js is:



An average of **1** version per month
28 contributors
33 releases
592 commits
14000+ lines of code
More than **120** files of code
More than **250** forks
A bandwidth of **1 TB** per month for the website
1.3GB (Code and samples)

How to use Babylon.js?

Open source project (Available on Github)

http://www.babylonjs.com

https://github.com/babylonjs/babylon.js

How to use it? **Include** one file and you're ready to go!

```
<script src="babylon.js"></script>
```

To start Babylon.js, you've just need to create an **engine** object:

```
var engine = new BABYLON.Engine(canvas, true);
```

How to use Babylon.js?

Babylon.js is a **scene graph**: All complex features are abstracted for **YOU**!

```
var scene = new BABYLON.Scene(engine);

var camera = new BABYLON.FreeCamera("Camera", new BABYLON.Vector3(0, 0, -10), scene);
var light0 = new BABYLON.PointLight("Omni0", new BABYLON.Vector3(0, 100, 100), scene);
var sphere = BABYLON.Mesh.createSphere("Sphere", 16, 3, scene);
```

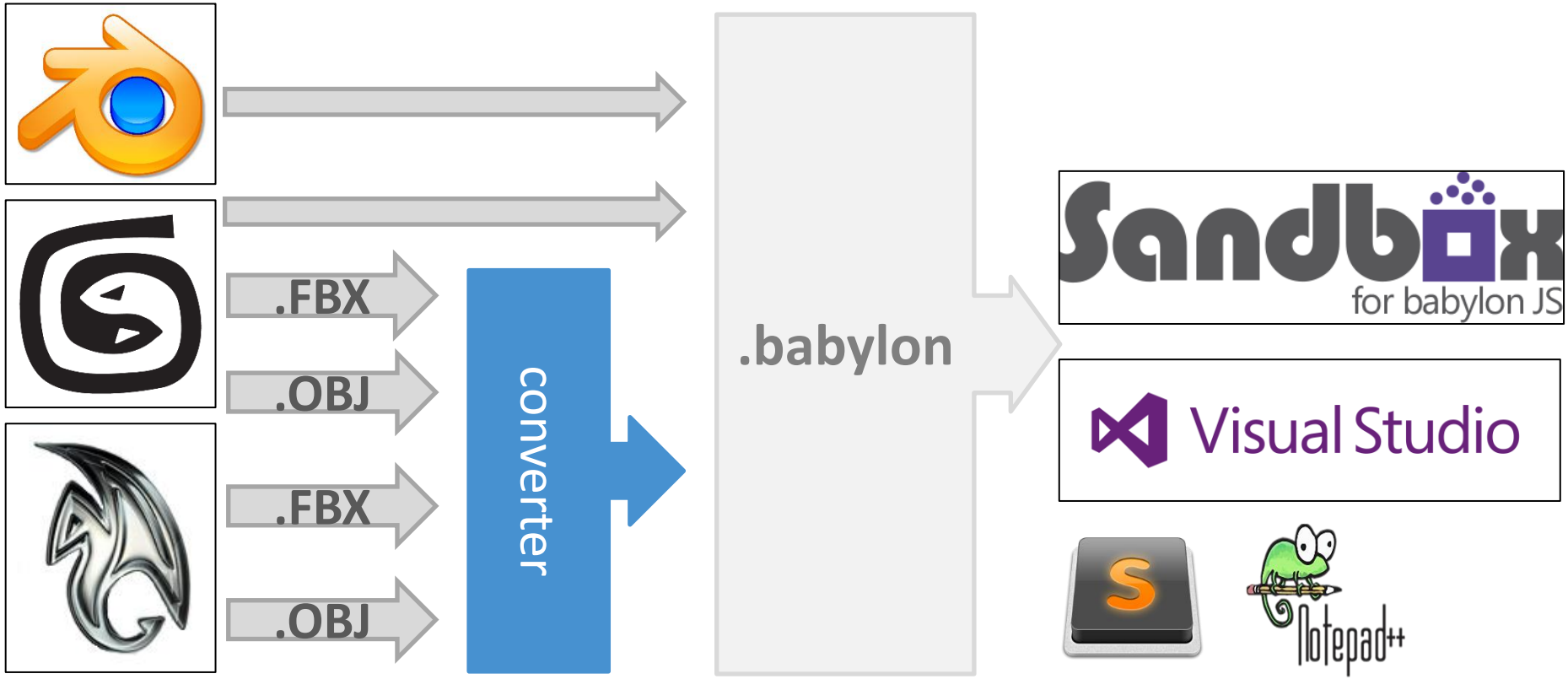
Handling **rendering** can be done in one line:

```
engine.runRenderLoop(function() { scene.render(); });
```

Did you say features?

Complete scene graph with lights, cameras, materials and meshes
Collisions engine
Physics engine (thanks to cannon.js)
Scene picking
Antialiasing
Animations engine
Particles Systems
Sprites and 2D layers
Frustum clipping
Sub-meshes clipping
Hardware scaling
Selection octrees
Offline mode (Assets are saved locally to prevent reloading them)
Incremental loading
Hardware accelerated instances
Diffuse lightning and texture
Ambient lightning and texture
Specular lightning
Opacity texture
Reflection texture (Spheric, planar, cubic and projection)
Mirror texture
Emissive texture
Specular texture
Bump texture
Up to 4 lights (points, directionals, spots, hemispherics)
Custom materials
Skybox
Vertex color
4 bones per vertex
Fog
Alpha blending
Alpha testing
Billboarding
Fullscreen mode
Shadow Maps and Variance Shadow Maps
Rendering layers
Post-processes (blur, refraction, black'n'white, fxaa, customs...)
Lens flares
Multi-views
Render target textures
Dynamic textures (canvas)
Video textures
Compressed (DDS) textures
Arc rotate camera
Free camera
Touch camera
Virtual Joysticks camera
Oculus Rift camera
Gamepad camera
Mesh cloning
Dynamic meshes
Height maps
Bones
Constructive solid geometries
Babylon scene file can be converted from *.OBJ*, *.FBX*, *.MXB*
Exporter for Blender
Exporter for Cheetah3d
Exporter for 3ds Max
Support for drag'n'drop

Creation Pipeline



Blender to Babylon.js exporter features support

Cameras

- Name
- Position
- Target
- Fov
- Clip start
- Clip end
- Check collisions
- Gravity
- Ellipsoid

Lights

- Type (Point, directional (Sun), Spot, Hemispheric)
- Name
- Position
- Direction
- Spot size
- Spot blend
- Energy
- Diffuse color
- Specular color

Materials & Multi-mat

- Name
- Ambient color
- Diffuse color
- Specular color
- Specular hardness
- Emissive color
- Alpha
- Backface culling
- Diffuse texture
- Ambient texture
- Opacity texture
- Reflection texture
- Emissive texture
- Bump texture

Textures

- Name
- Associated file
- Level
- Use alpha
- uOffset / voffset
- uScale / uScale
- uAng / vAng / Wang
- WrapU / WrapV
- Coordinates index

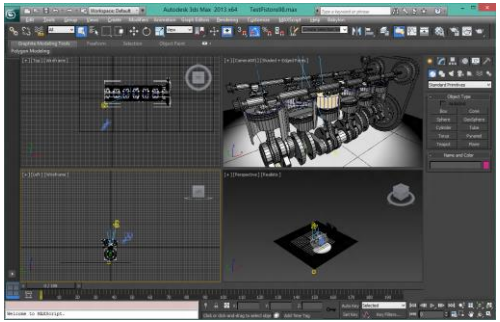
Meshes

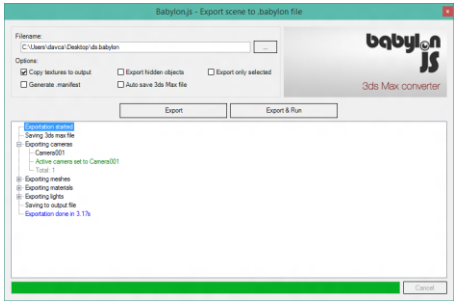
- Name
- Geometry (Positions & normals)
- Position
- Rotation
- Scaling
- Texture coordinates (2 channels)
- Vertex colors
- Visibility
- Check collisions
- Billboard
- Receive and cast shadows
- Bones (armatures) and bones' animations
- ◦Animations

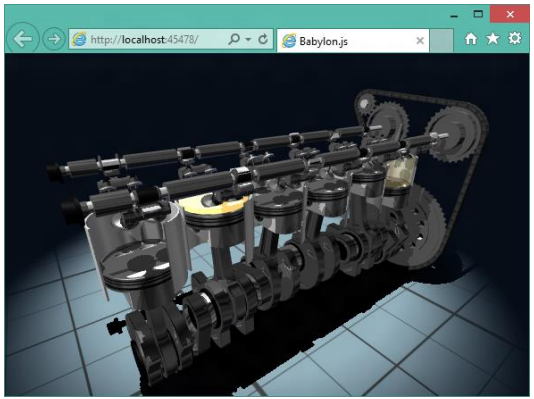
Fully integrated pipeline exportation

- **One click** exportation
- Integrated **web server**

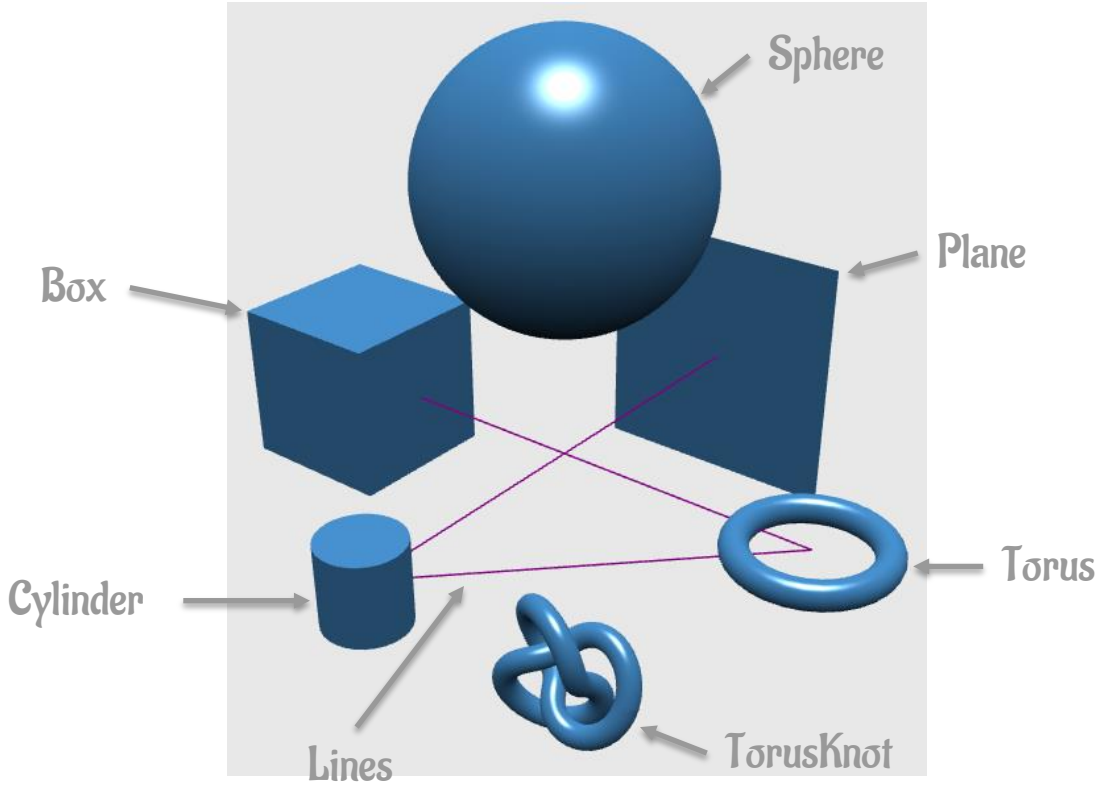
- Export:
  - Cameras, lights, meshes
  - Animations and regular materials



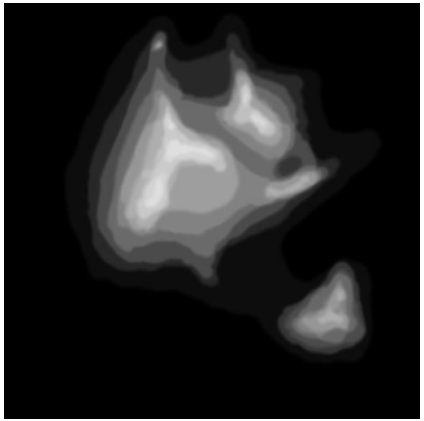
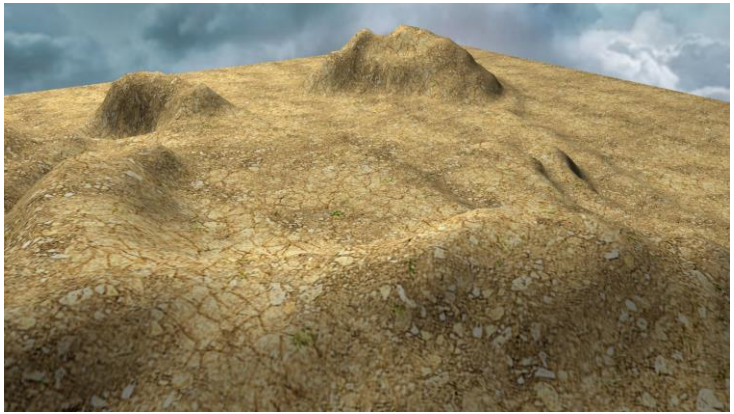




BABYLON.JS Meshes main primitives

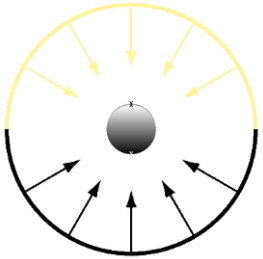


Ground & Ground From HeightMap

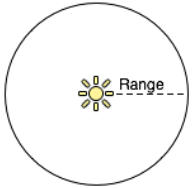

+

=


BABYLON.JS Lights

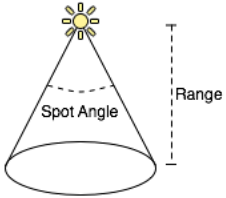
HemisphericLight



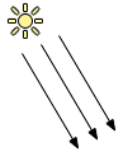
PointLight

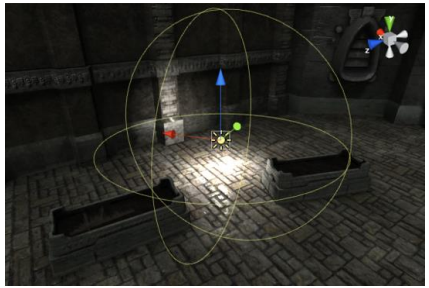


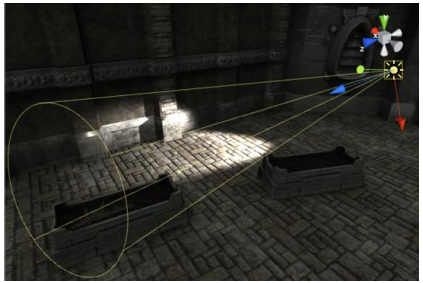
SpotLight

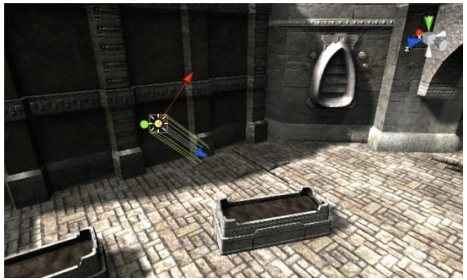


DirectionalLight



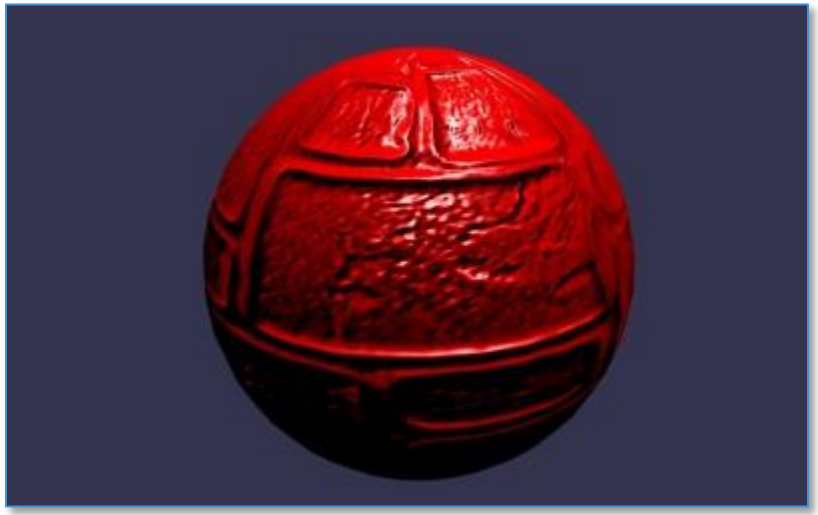






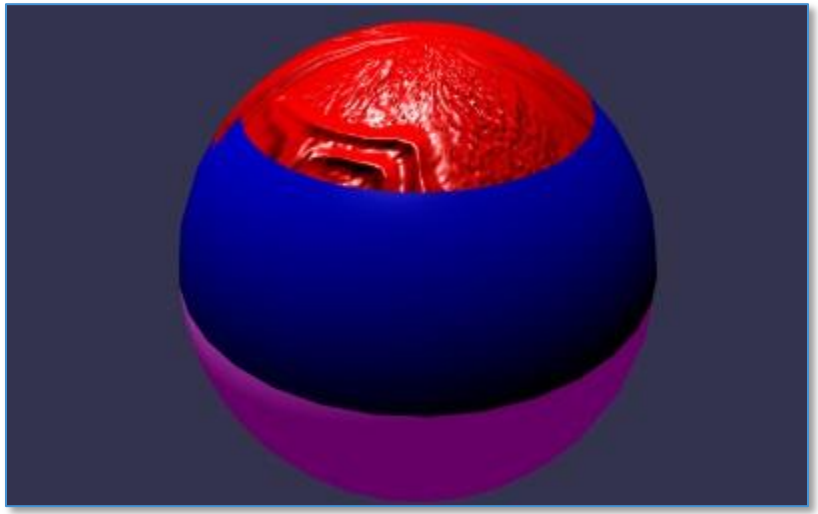
Uber shader - StandardMaterial

- StandardMaterial hides shaders **complexity**
- It supports:
  - Diffuse
  - Specular
  - Emissive
  - Bump
  - Opacity
  - Ambient
  - Reflection
  - Alpha



MultiMaterial

- Container allowing you to apply many materials to a single object
- Segment a mesh into **sub-meshes**



Dynamic texture

- What about using **canvas 2D API** to generate texture content?
- This is the magic done by **BABYLON.DynamicTexture**
- All 2D API are available (Text, Arc, etc..)

- **Warning:** each update will trigger a copy to GPU memory

PLAYING WITH **INPUT**

Touch

Camera based on **pointer events**

Device Orientation

Camera based on **Device Orientation API**

Virtual Joysticks

Using pointer events, this camera generates **two joysticks** on top of the scene

Anaglyph

Use this camera with **Red/Green** glasses

Oculus

Control camera orientation with **Oculus Rift** device

Gamepad

Use your **gamepad** to control your camera

Learning Babylon.js using the playground

- Get sample code
- Try and experiment
- Share with friends
- Learn by reading examples