VGA
Web Graphics

Ivo Pisařovic

MENDELU
Faculty
of Business
and Economics

Outline

1. Motivation to learn Javascript
2. Classic WebGL
3. A-Frame

PART 1
Motivation to learn Javascript

1. **Motivation to learn Javascript**
2. Classic WebGL
3. Three.js
4. A-Frame

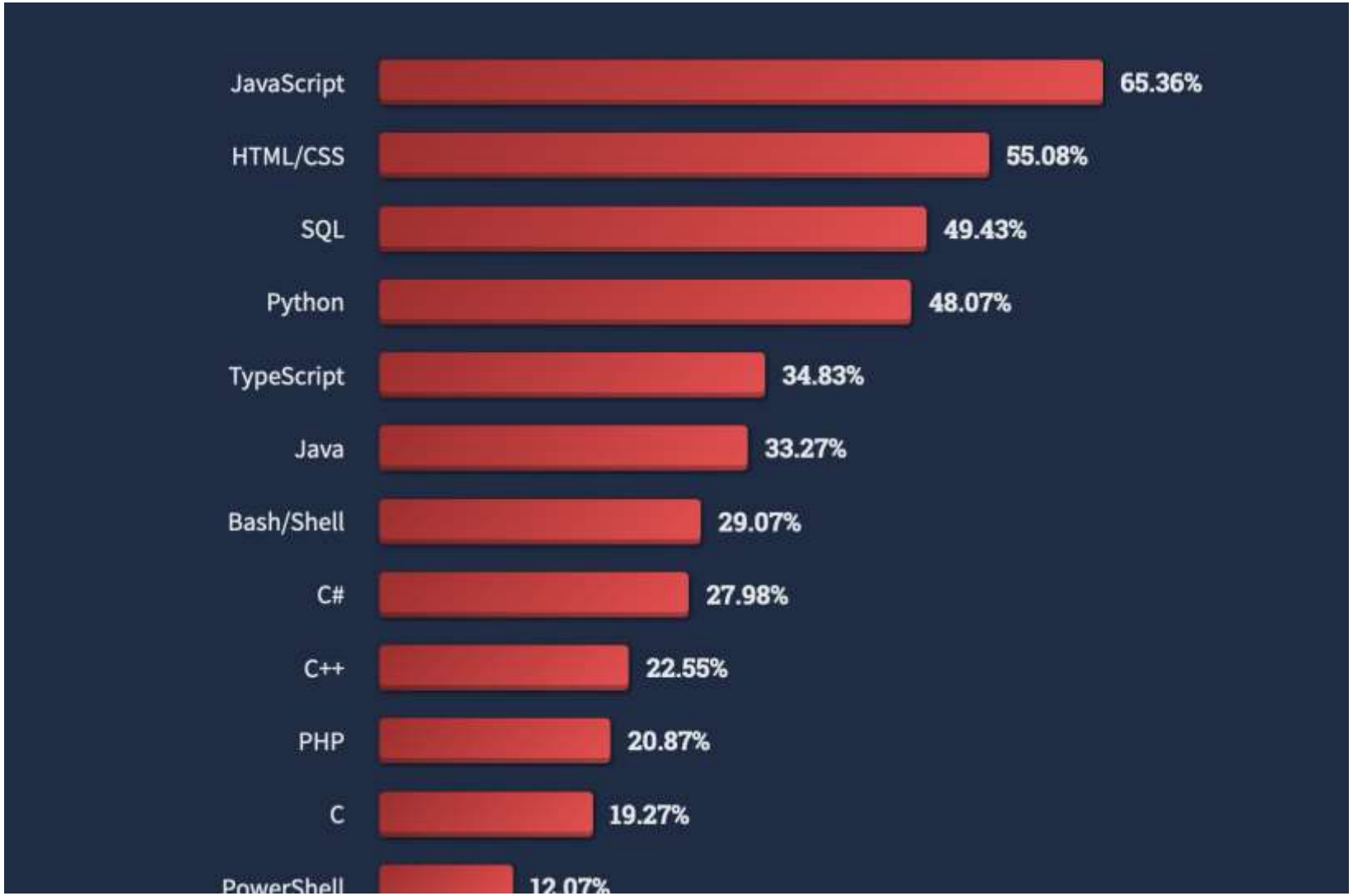
`"b" + "a" + + "a" + "a"`	`"baNaNa"`
`"" == false`	`true`
`0 == false`	`true`
`true + true + true`	`3`

`(![] + [])[+[]] +`
`(![] + [])[+!+[]] +`
`([![]] + [][[]])[+!+[] + [+[]]] +`
`(![] + [])[!+[] + !+[]];`
	`"fail"`

https://github.com/denysdovhan/wtfjs#true-is-false

So why do ANYthing in Javascript?

Language popularity = number of jobs



https://insights.stackoverflow.com/survey

MENDELU
Faculty
of Business
and Economics

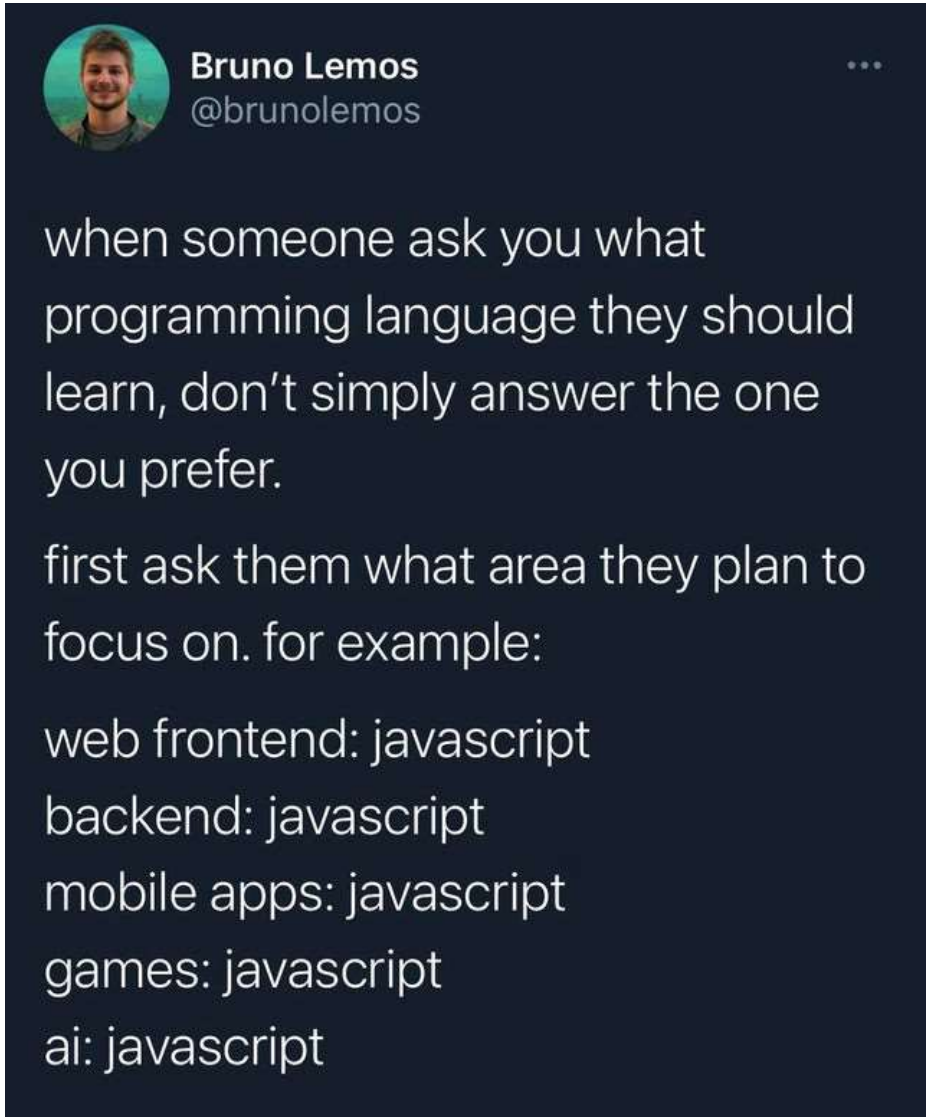
Cross-platform

One developer, one app, all platforms

Significant cost reduction

Monorepo approach (BE+FE together)

- Web browser
- Desktop (Windows, macOS, Linux, ...)
- Mobile (Android, iOS)
- Virtual Reality (HTC Vive, Oculus, ...)
- Augmented reality (Microsoft HoloLens)
- Machine learning (TensorFlow.js)



Modern syntax (ECMAScript standard)

Arrow functions

```
const visibleObjects = objects.filter(object => object.isVisible)
```

Optional chaining

```
const sword = character?.inventory?.sword
```

Nullish coalescing

```
character.name ?? "Unknown name"
```

Async/await, promises

```
async function downloadModel() {
    const response = await fetch('https://example.com/model.json')
    this.model = await response.json()
}
```

Typescript 😍 = stable development with the power of javascript


USE JAVASCRIPT
YOU MUST.
imgflip.com

PART 2
Classic WebGL

1. Motivation to learn Javascript
2. **Classic WebGL**
3. Three.js
4. A-Frame

Ways to implement a 3D web app

a) **Vanilla** Javascript and WebGL
b) **Medium-level** libraries, e.g. Three.js, Babylon.js
c) **High-level** libraries, e.g. A-FRAME
d) 3D software, e.g. Blender and export
e) Game Engines, e.g. Unity or Unreal Engine and export for web

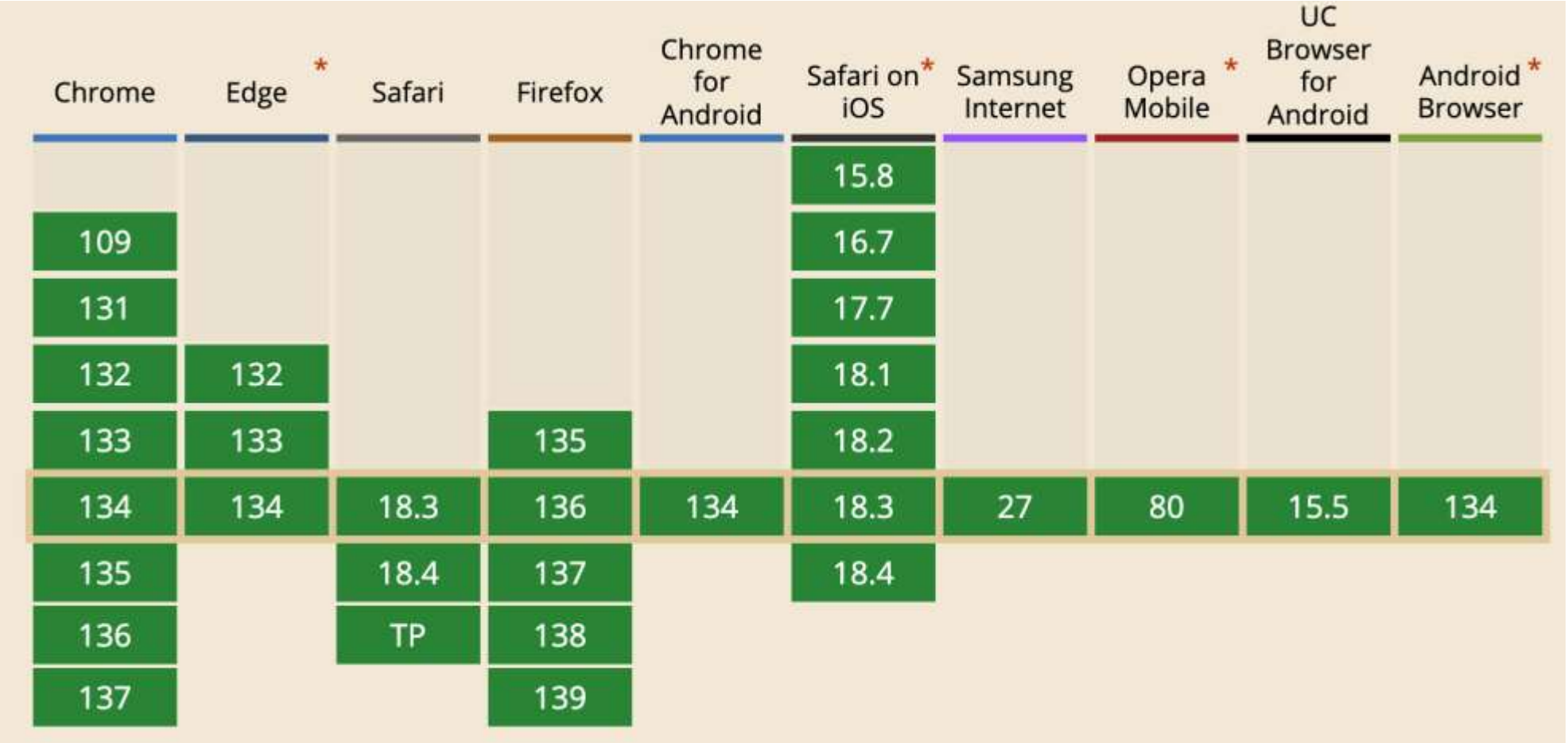
WebGL

- Javascript API for native graphic rendering in a web browser
- Uses HTML element <canvas>
- WebGL 1.0 based on OpenGL ES 2.0 (no fixed pipeline)
- **WebGL 2.0 based on OpenGL ES 3.0 (all browser supported)**

WebGPU

- New, more efficient technology
- Based on modern graphic APIs (Vulkan, Metal, Direct3D)
- Advanced shading and ray tracing, new GPU compute shaders
- Chrome, Edge supported; Safari, Firefox, etc. not supported yet!

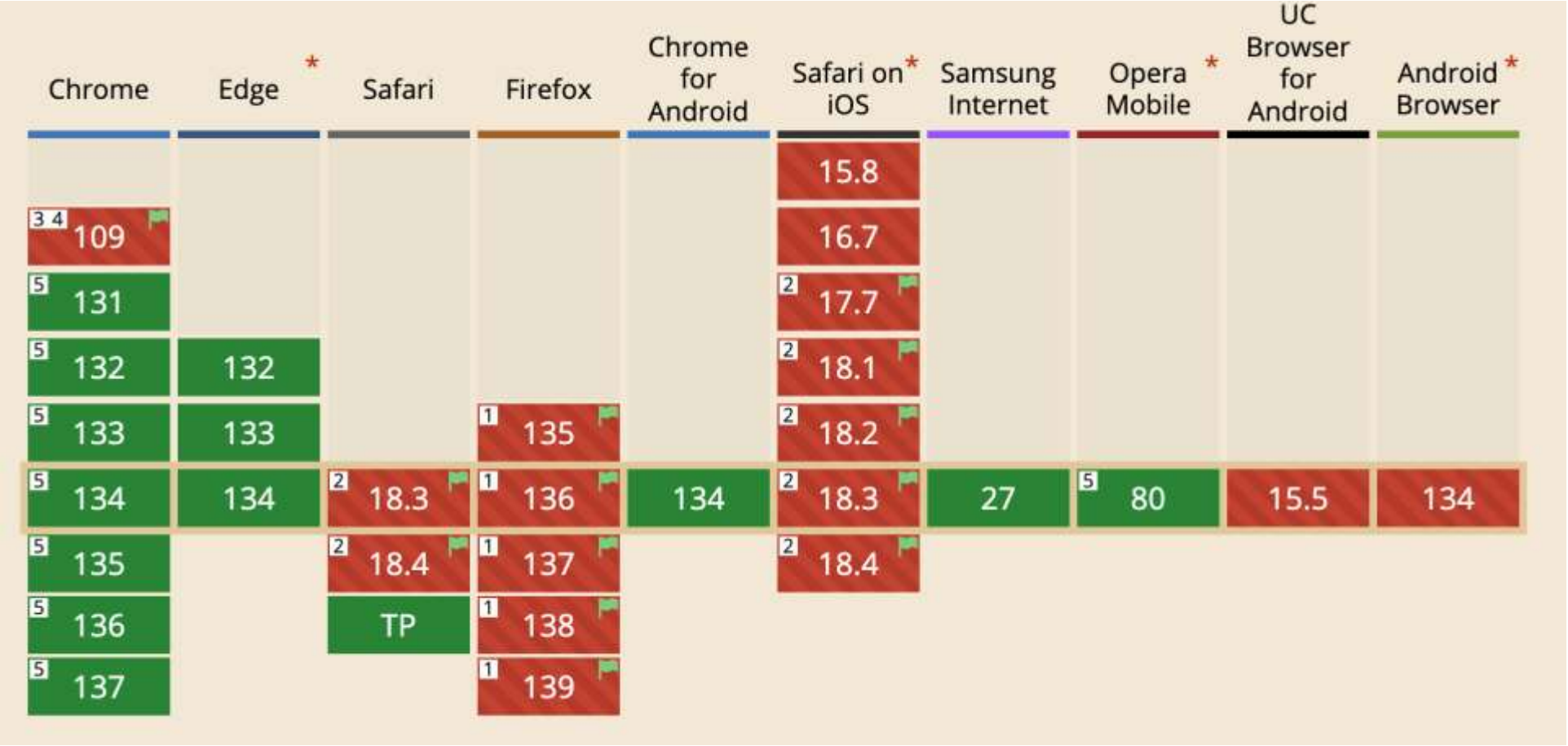
Support of WebGL in web browsers



Chrome	Edge *	Safari	Firefox	Chrome for Android	Safari on iOS *	Samsung Internet	Opera Mobile *	UC Browser for Android	Android Browser *
					15.8				
109					16.7				
131					17.7				
132	132				18.1				
133	133		135		18.2				
134	134	18.3	136	134	18.3	27	80	15.5	134
135		18.4	137		18.4				
136		TP	138						
137			139						

https://caniuse.com/?search=webgl

Support of WebGPU in web browsers



https://caniuse.com/?search=webgpu

MENDELU
Faculty
of Business
and Economics

Pure WebGL example: HTML part

```
<html>
  <body>
    <script id="vs" type="x-shader/x-vertex">
      attribute vec2 coordinates;
      void main(void) {
        gl_Position = vec4(coordinates,0.0, 1.0);
      }
    </script>
    <script id="fs" type="x-shader/x-fragment">
      void main(void) {
        gl_FragColor = vec4(0.0, 0.0, 0.0, 0.1);
      }
    </script>
    <canvas width="800" height="600" id="my_Canvas"></
    <script src="js/main.js" type="text/javascript"></
  </body>
</html>
```

Pure WebGL example: buffers

```
// definice vrcholu 3x [x,y]
var vertices = [-0.5, 0.5, -0.5, -0.5, 0.0, -0.5,];

// vytvoreni bufferu
var vertex_buffer = gl.createBuffer();

// bind bufferu
gl.bindBuffer(gl.ARRAY_BUFFER, vertex_buffer);

// nahrani dat
gl.bufferData(gl.ARRAY_BUFFER,
  new Float32Array(vertices), gl.STATIC_DRAW);

// odbindovani bufferu
gl.bindBuffer(gl.ARRAY_BUFFER, null);
```


TONIGHT WE DINE IN
WEBGHELL

PART 2
Three.js

1. Motivation to learn Javascript
2. Classic WebGL
3. **Three.js**
4. A-Frame

Three.js

Middle-level wrapper over WebGL

Common things are ready-to-use objects (cubes, lights, camera, …).

Imperative OOP syntax

Example of creating a 3D box with the given size and color:

```
const geometry = new THREE.BoxGeometry( 1, 1, 1 );
const material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
const cube = new THREE.Mesh( geometry, material );
scene.add( cube );

cube.position.z = 5;
cube.rotation.y = Math.PI / 2;
```

Three.js

Example of loading a model and adding it to the scene:

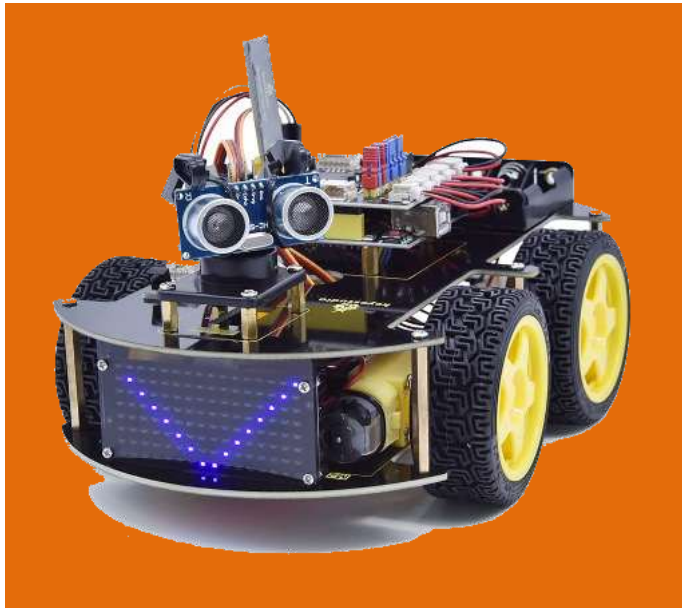
```
// Instantiate a loader
const loader = new GLTFLoader();

// Load a glTF resource
loader.load(
    'models/gltf/duck/duck.gltf',
    function (gltf) { // called when the resource is loaded
        scene.add(gltf.scene); // gltf may contains also camera, lights,...
        gltf.scene.position.set(3, -4, 5);
    },
    function (xhr) { // called while loading is progressing
        console.log((xhr.loaded / xhr.total * 100) + '% loaded');
    },
    function (error) { // called when loading has errors
        console.log('An error happened');
    }
);
```

PART 3
A-Frame

1. Motivation to learn Javascript
2. Classic WebGL
3. Three.js
4. **A-Frame**

What are the key factors of innovations?

Did you know that the light bulb was developed independently by several scientists? Edison was about 10-20th in order. Why didn't the first light bulb become famous?
The technology must be cheap and accessible for everyone to try. One must establish an environment that doesn't inhibit experimentation.



How to accelerate innovations in 3D development?

Build on a technology that everyone knows.

HTML a open-source webové technologie

Enable to run everywhere.

Podpora různých AR helem a hlavně mobilů

No need of special expensive hardware.

Možnost vyvíjet zdarma na jakémkoliv počítači

MENDELU

A-FRAME

- High-level wrapper over Three.js
- Zero-code support for VR and AR
- Component architecture
- Cross-platform (web, mobile, Meta Quest, HTC Vive, Microsoft Holo Lens)
- Declarative syntax (using HTML to define scene)

```
<a-scene>
    <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
    <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
    <a-cylinder position="1 0.75 -3" radius=”2" height=”3" color="#FFC65D"></a-cylinder>
    <a-plane position="0 0 -4" rotation="-90 0 0" width="4" height="4”></a-plane>
    <a-sky color="#ECECEC"></a-sky>
</a-scene>
```

A-FRAME Device Support

HTC Vive Pro
Meta Quest 3
Meta Quest Pro
Microsoft HoloLens 2
iPhone Pro
iPad Pro
...

A-FRAME Architecture

A-FRAME Entities

Everything is a general ***entity****:*

```
<a-entity geometry="primitive: box; width: 3" material="color: red"></a-entity>
```

Syntactic helpers for common objects = ***primitives***:

```
<a-sky color="#ECECEC"></a-sky>
<a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
<a-box color="red" width="3"></a-box>
```

A-FRAME Components

- Entity-component-system instead of inheritance and hierarchy (OOP, three.js)
- Common in game and VR/AR development
  - More suitable than OOP because there are many objects that must interact with each other. Using OOP would lead to a dependency hell.
- A set of reusable encapsulated modules attached to entities

System
`<a-scene>`

Entity
`<a-entity>`

Component
`material="color: red"`

Component
`grab`

Component
`physics="mass: 0.78"`

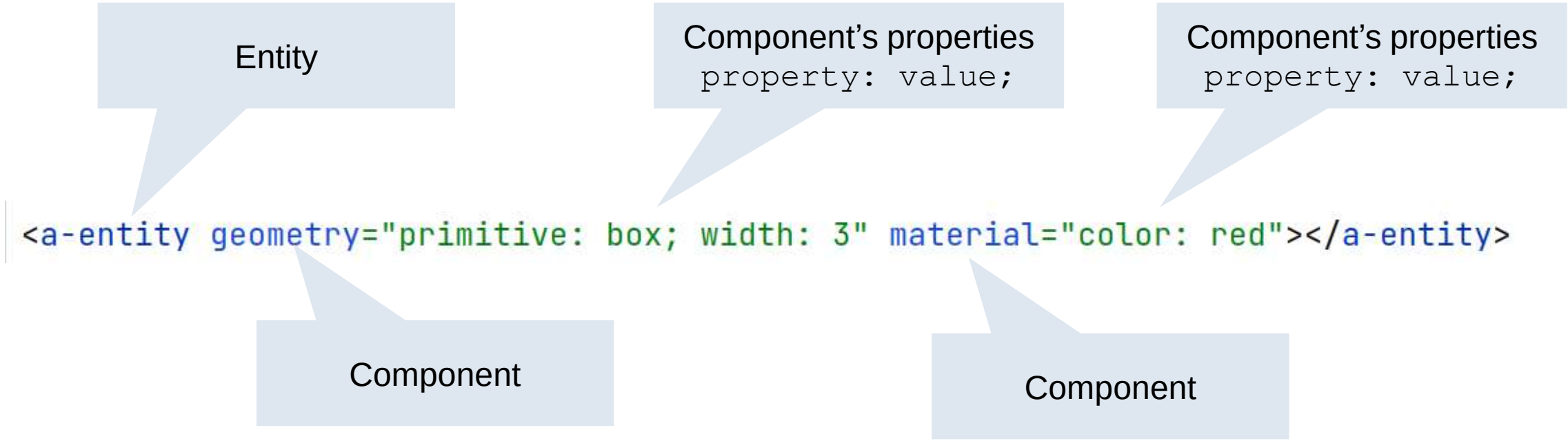
Component
`oculus-touch-controls`

Entity
`<a-entity>`

Component
`grab`

Component
`material="color: red"`

A-FRAME Components



MENDELU
Faculty
of Business
and Economics

A-FRAME Component Example

```
AFRAME.registerComponent('player', {
    schema: {
        role: {default: 'magician'}
    },
    init: function () {
        // magician will have a violet aura
        let color = this.data.role === "magician" ? '#3322a3' : '#376b27';

        const el = this.el;  // <a-entity>
        el.setAttribute( qualifiedName: 'color', color); // <a-entity color="#3322a3">
    }
});
```

```
<a-entity player="role: soldier"/>
```

MENDELU
Faculty
of Business
and Economics

A-FRAME Events

```
<a-entity
  gltf-model="baby-yoda.gltf"
  have-force
></a-entity>
```

Component: gltf-model

Component: have-force

```
emit('force-event')
```

Component ”have-force” attached to entity baby-yoda just emited an event “force-event” that will be caught by the rock, causing it to fly.

```
<a-entity
  gltf-model="rock.gltf”
></a-entity>
```

Component: gltf-model

MENDELU
Faculty
of Business
and Economics

Comparison of approaches to rendering content

Imperative approach

Function-based
Manual UI updates through method calls
Vanilla JS, WebGL, Three.js

Reactive approach

Data-driven
UI updates automatically based on changes in data
Vue, React, Qt

Event based system

Event-based
Emiting events to trigger an update
A-FRAME

For nerds:
It is very beneficial to combine Vue or React with A-FRAME and reactively update 3D scene ☺

MENDELU
Faculty
of Business
and Economics

Summary of Web Graphics

Pros:

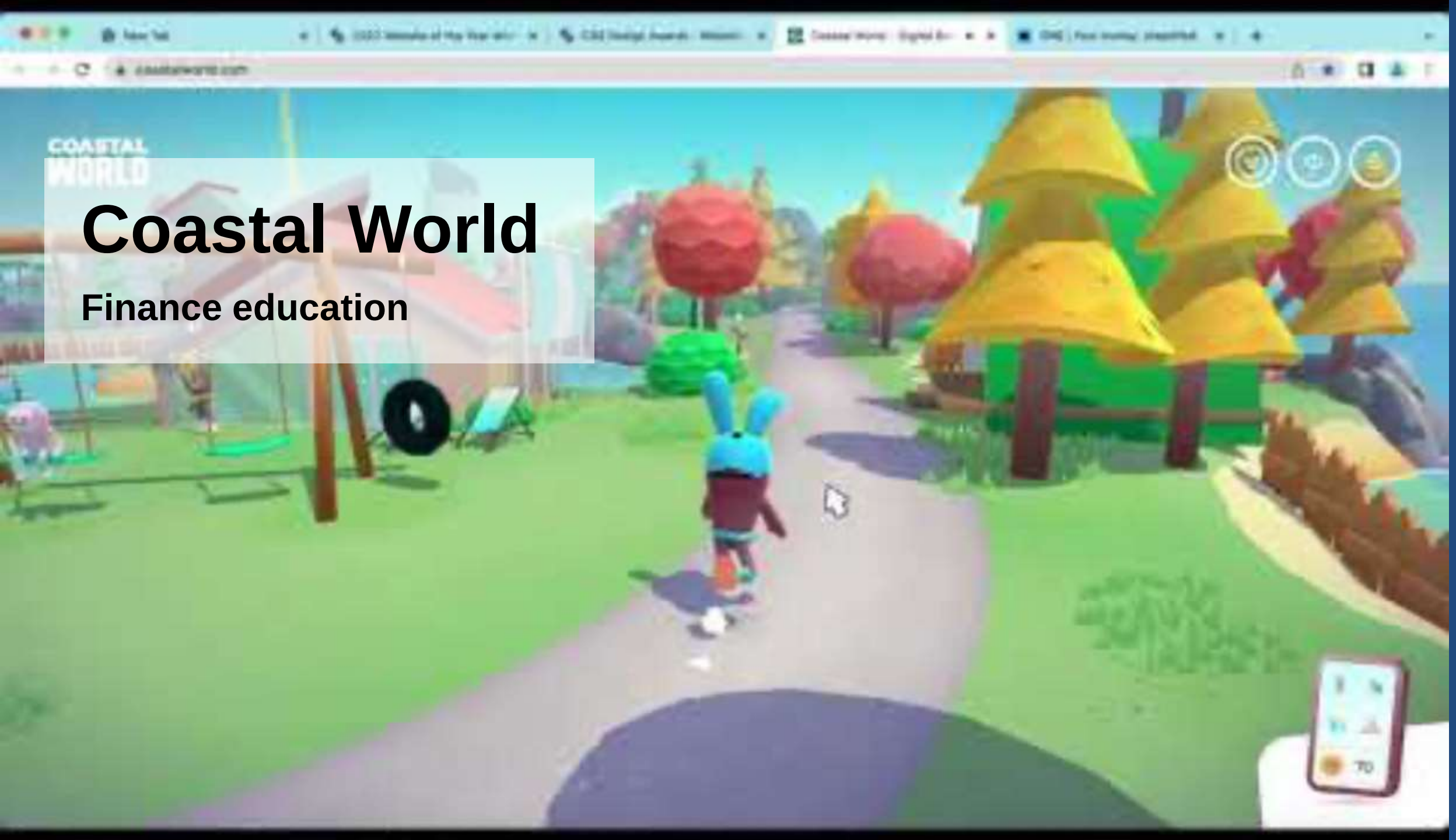
- Quick development & prototyping
- Cross-platform by design
- Large community
- Open technologies
- Easily distributed to users via a URL

Cons:

- Everything must be dowloaded when the user load the page => low quality models
- Less performance
  (although WebGPU instead brings improvements over WebGL)
- Absence of complex game engines, more things you must create from scratch

Ukázky z praxe

- **Interland**
- **Online safety by Google**



Coastal World

Finance education


Scavenger Land
Tactical battle with NFTs

Q&A

MENDELU
Faculty
of Business
and Economics

The 90s:
Don’t sit too close to the TV

Now:

PG2

Web Graphics: Animation

Ivo Pisařovic

MENDELU
Faculty
of Business
and Economics

Outline

1. Disney’s 12 principles of animation
2. Preparing a scene in Blender
3. Simple animation in Blender
4. Custom character

PART 1
Disney’s 12 principles of animation

1. **Disney’s 12 principles of animation**
2. Preparing a scene in Blender
3. Simple animation in Blender
4. Custom character

MENDELU
Faculty
of Business
and Economics

Disney’s principles of animation

- Ollie Johnston and Frank Thomas, 1981
- Work of Disney animators since 1930
- Useful for animating characters in movies and games but also web and mobile apps
- Giving animated objects feel of natural living organism
- Making animation fun and entertaining
- Intentionally not realistic

12 principles

1. Squash and stretch
2. Anticipation
3. Staging
4. Straight ahead action and pose to pose
5. Follow through and overlapping action
6. Slow in and slow out
6. Arc
7. Secondary action
8. Timing
9. Exaggeration
10. Solid drawing
11. Appeal

Squash and stretch

- The most important principle
- Illusion of gravity, weight, flexibility
- E.g. a bouncing rubber ball

Anticipation

- Preparing for an action
- A car starts from one place lineary accelerating (inefficient)
- Living organism use physics to help them move
- E.g. you need to bend knees before jumping

Staging

- Use motion to guide the eyes
- Draw attention
- Add supporting elements to emphasize motion (arrows, glow, color)

Arc + Slow-in and slow-out

- Respecting physics
- Objects are usually not moving lineary in real world
- Use a curve for easing
- Use a curve for trajectory

Break physics
Break rules
But BE CONSISTENT

See all principles here:

https://www.creativebloq.com/advice/understand-the-12-principles-of-animation

MENDELU PEF

PART 2
Preparing a scene in Blender

1. Disney’s 12 principles of animation
2. **Preparing a scene in Blender**
3. Simple animation in Blender
4. Custom character

Steps

1. Design your scene in a 3D software
2. Export into GLTF / OBJ (+ textures)
3. Import into A-Frame using an asset and GLTF loader
4. Fix scaling and rotation

PART 3
Simple animation in Blender

1. Disney’s 12 principles of animation
2. Preparing a scene in Blender
3. **Simple animation in Blender**
4. Custom character

Steps

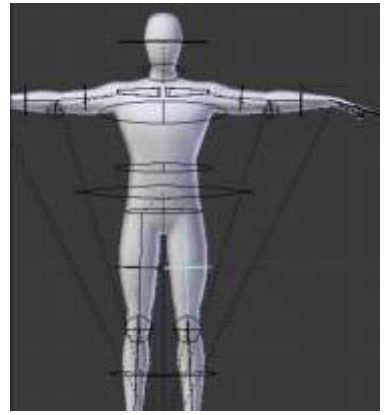
1. Prepare and fine-tune your object before animating
2. Add initial keyframe
3. Change position of vertices and add the final keyframe
4. Optionally add intermediate keys to have non-linear animation
5. Add names to your animations (in Action Editor)
6. Export GLTF, import into scene
7. Play animation using animation mixer in A-Frame Extras

PART 4
Custom character

1. Disney’s 12 principles of animation
2. Preparing a scene in Blender
3. Simple animation in Blender
4. **Custom character**

Steps

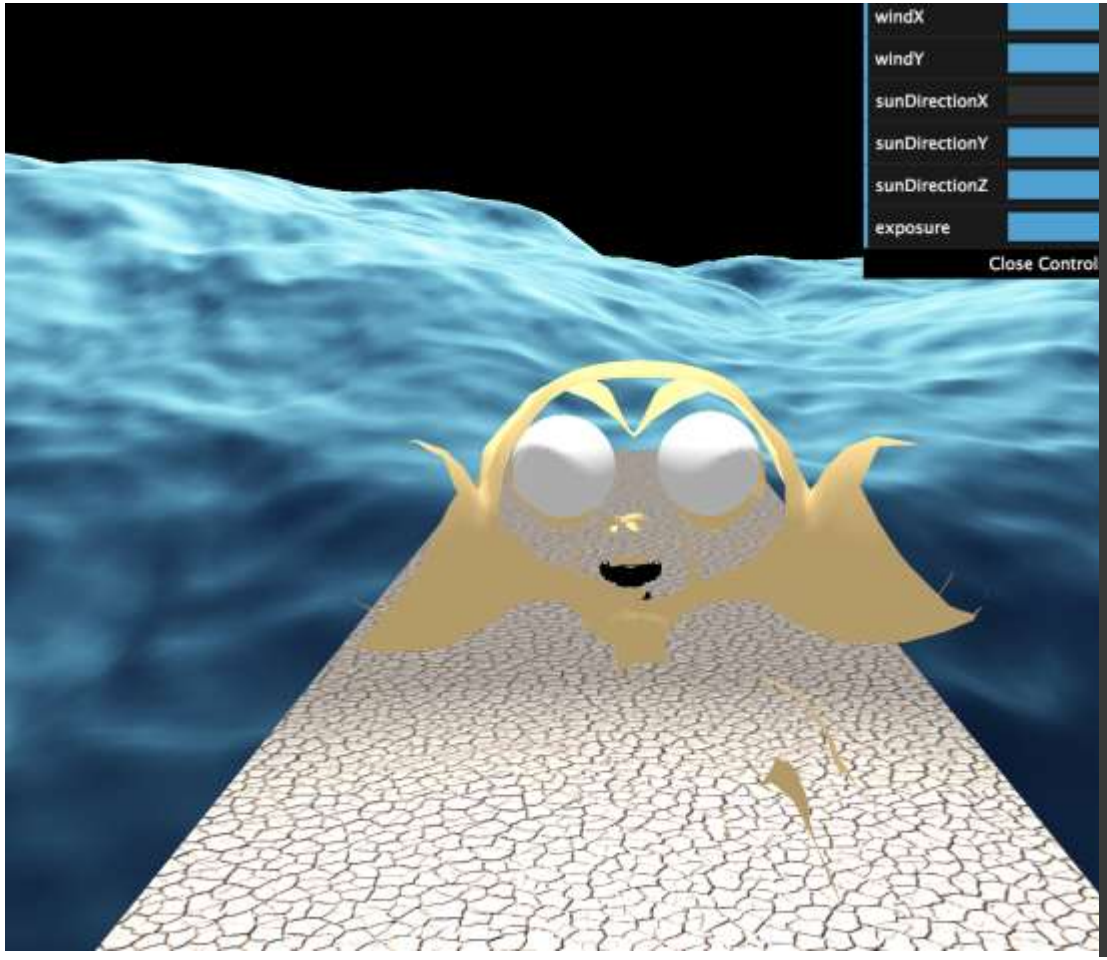
1. Prepare a figure with bones (it is easier to modify an existing one than creating from scratch)
2. Create animation like before, but move only bones
3. Meshes are automatically stretched/deformed to match the bones
4. Return to first keyframe and the default `pose` position
5. Export in Blender and load animations in A-Frame
6. Use fading for real transitions between animations

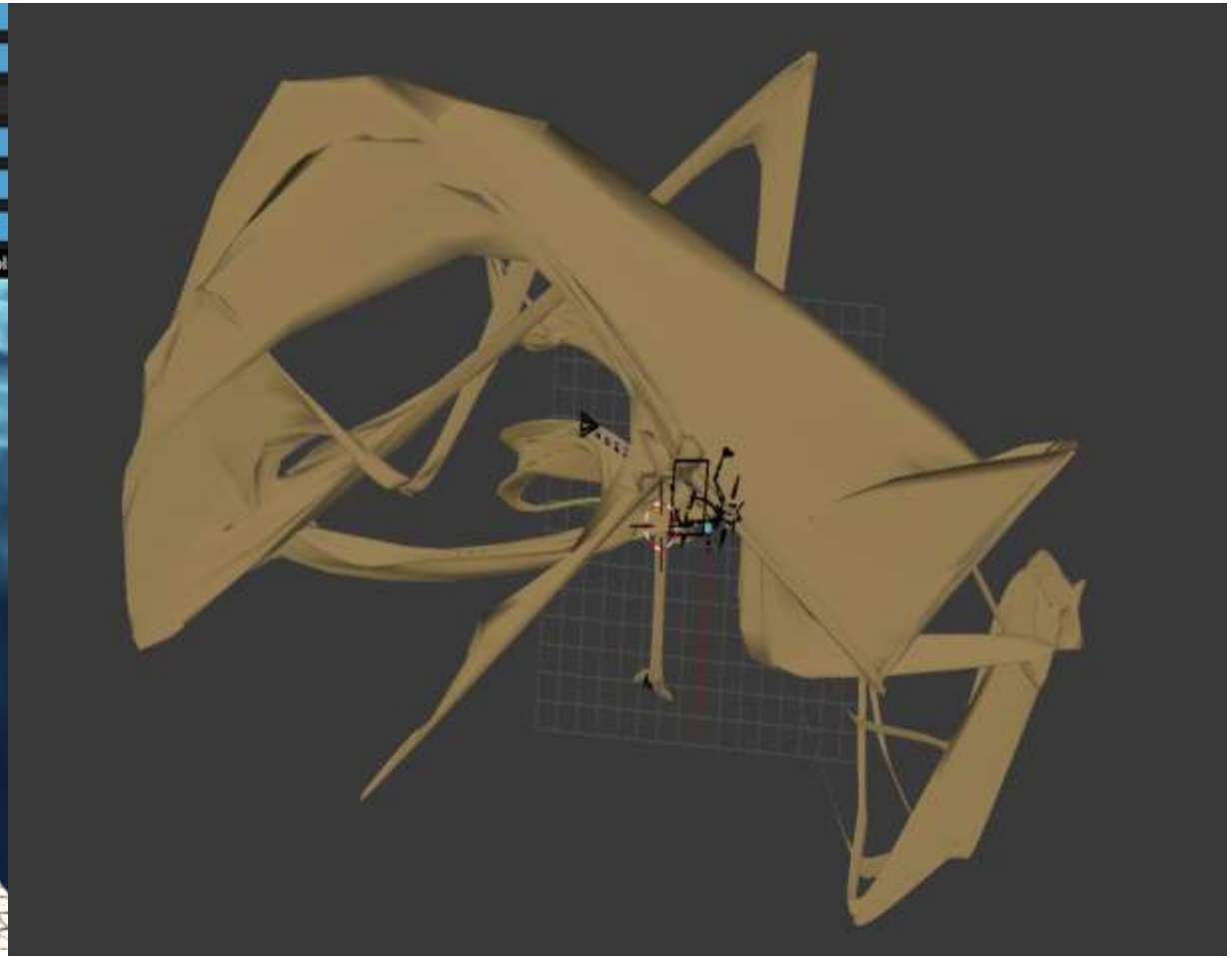


WARNING: take care of correctly selecting elements while animating, select bones not the object itself

WARNING: ensure you are editing the correct animation clip

Take care to prevent nightmares





https://youtu.be/1eLqfRYLFW8

MENDELU
Faculty
of Business
and Economics

Links

A complex tutorial from sketching your character, modelling, animating, texturing and exporting:

http://unboring.net/workflows/animation.html

UV texture mapping in Blender, how to add a texture to a complicated mesh like a character:

https://cgi.tutsplus.com/tutorials/creating-a-low-poly-ninja-game-character-using-blender-part-2--cg-16133

Q&A

MENDELU
Faculty
of Business
and Economics

VGA
Web Graphics

Ivo Pisařovic

Lightning Textures

MENDELU
Faculty
of Business
and Economics

Outline

1. Light
2. Shadow
3. Textures

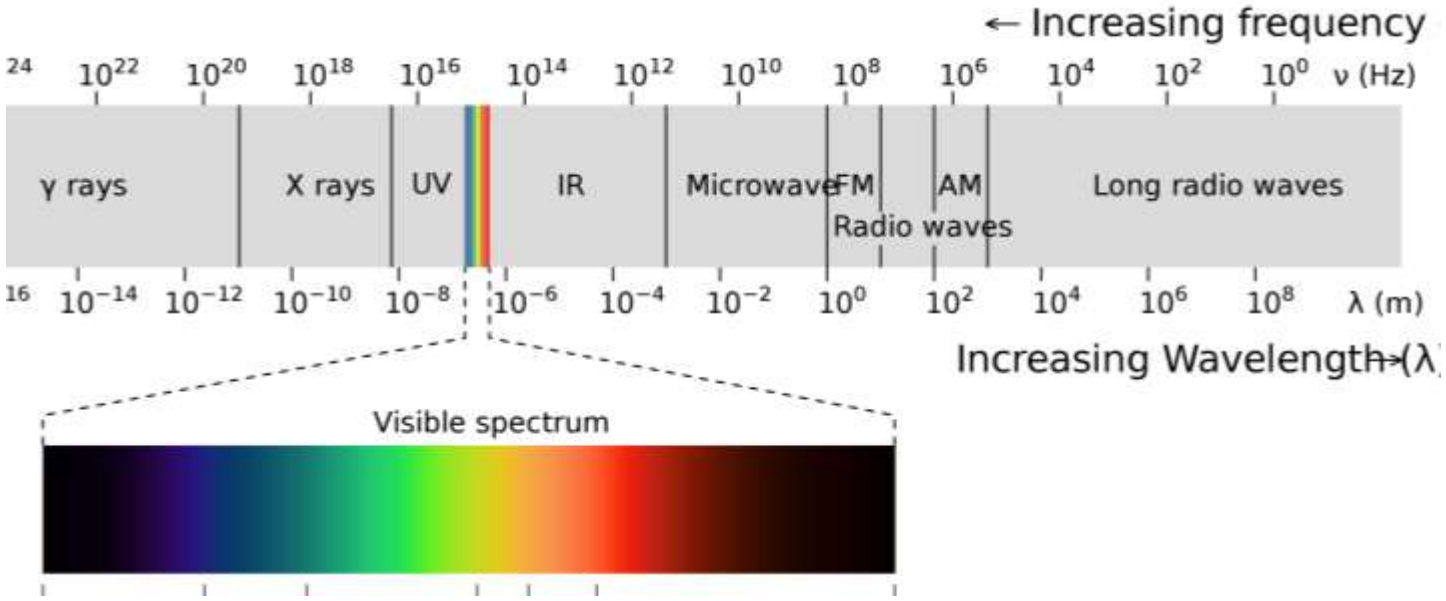
Light

1. **Light**
2. Shadow
3. Texture

Light

Light or visible light is electromagnetic radiation that can be perceived by the human eye.

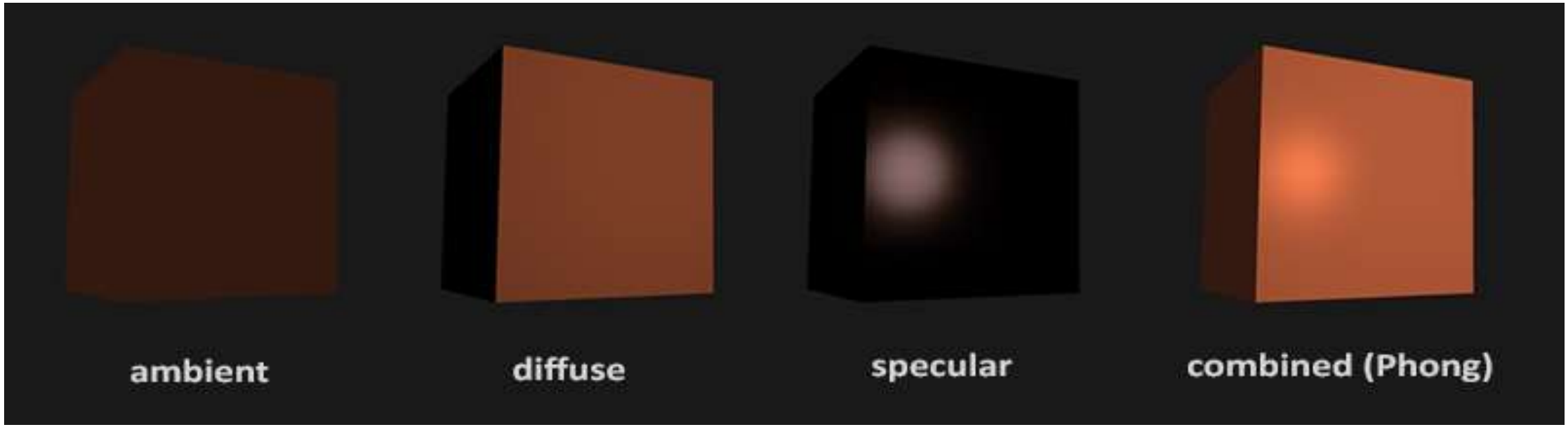
A mathematical function describing the intensity of a beam of scattered light as a function of its direction and the direction, intensity and wavelength of the incident beam. The models are physical (Torrance-Sparrow) or simplified (Phong illumination model).



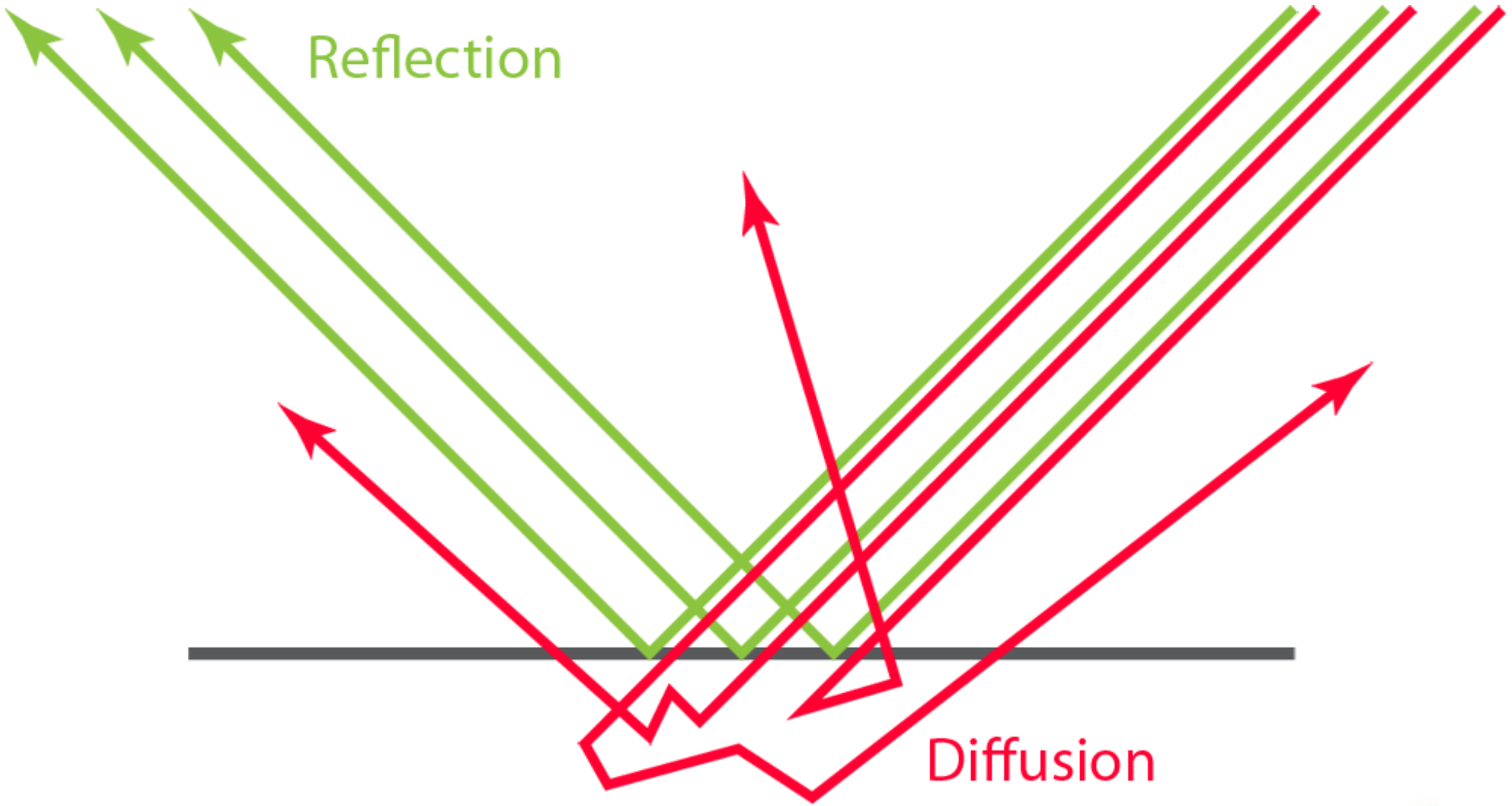
https://en.wikipedia.org/wiki/Light
https://en.wikipedia.org/wiki/Light#/media/File:EM_spectrum.svg

MENDELU
Faculty
of Business
and Economics

Light Components



http://hucoco.com/2018/07/09/Lighting_and_Materials/



https://marmoset.co/posts/basic-theory-of-physically-based-rendering/

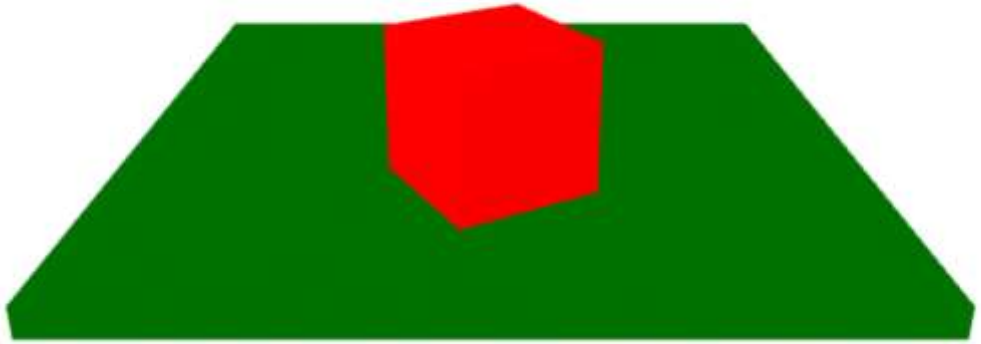
Light Components / Ambient

The very basic type and the most simple one

Is present in all scenes as nothing is completely dark

Simple to calculate but not realistic (use it only with low intensity)

```
float ambientStrength = 0.1;
vec3 ambient = ambientStrength * lightColor;
vec3 result = ambient * objectColor;
FragColor = vec4(result, 1.0);
```



Light Components / Diffuse

Comes from one direction

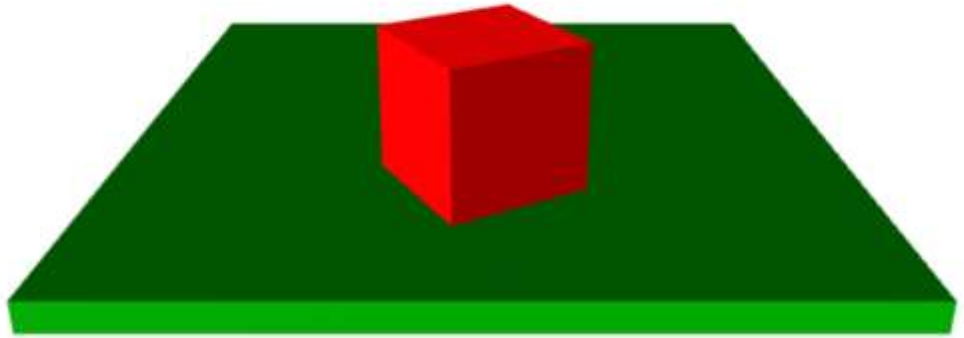
Reflects to all directions

Plastic feeling

Light goes under the surface and is scattered

The scattering is often so uniformly chaotic that it can be said to appear the same from all directions

```
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * lightColor;
vec3 result = diffuse * objectColor;
FragColor = vec4(result, 1.0);
```



Light Components / Specular

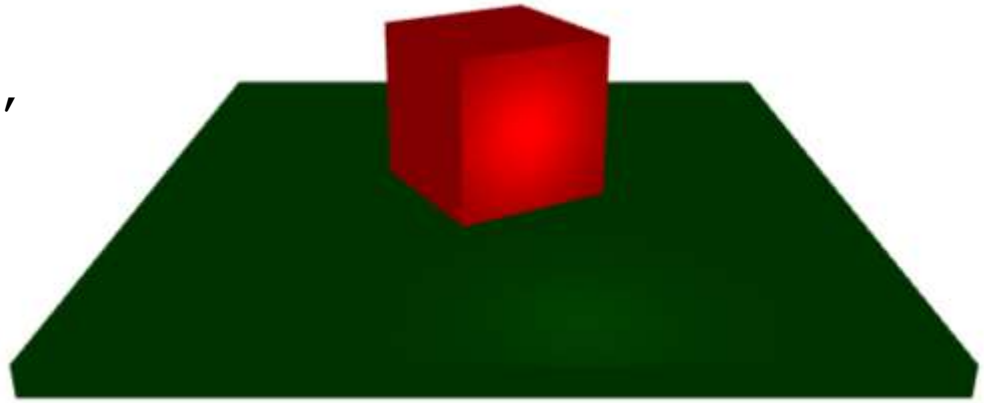
Comes from one direction

Reflects to one direction

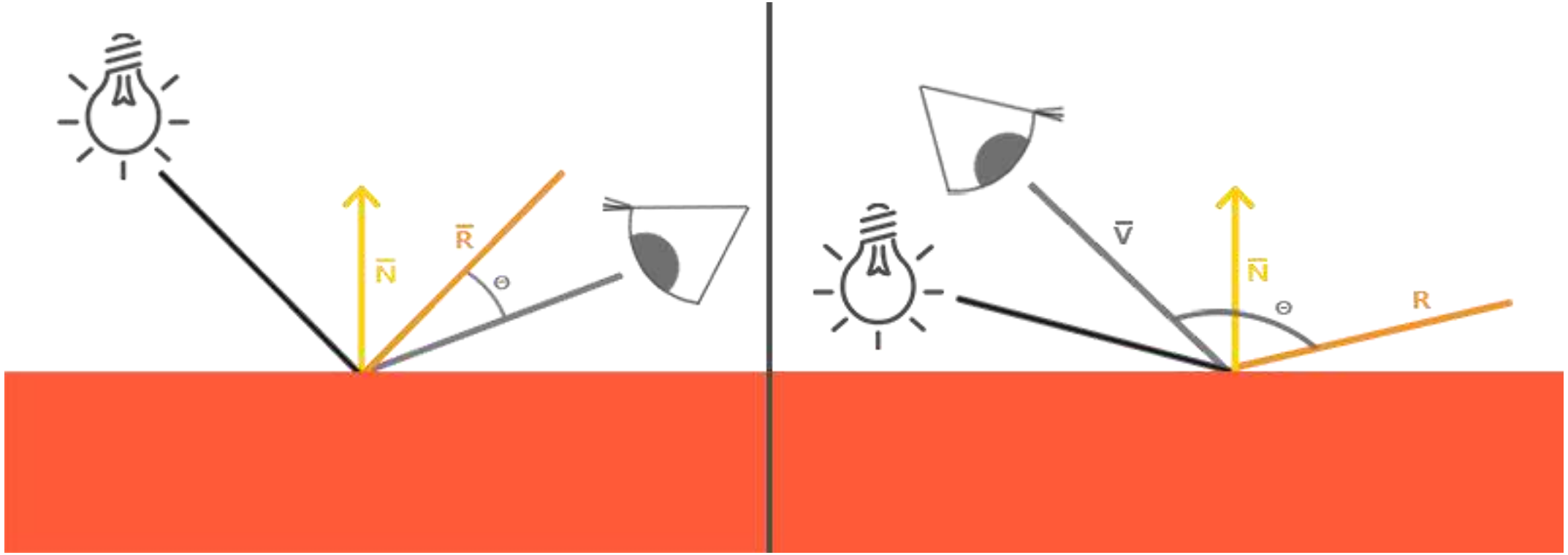
Metalic feeling

The visible intensity is determined by the angle between the eye and the direct reflected ray.

```
float specularStrength = 0.5;
vec3 viewDir = normalize(viewPos - FragPos);
vec3 reflectDir = reflect(-lightDir, norm);
float spec = pow(max(dot(viewDir, reflectDir),
  0.0), 32);
vec3 specular =
  specularStrength * spec * lightColor;
vec3 result = specular * objectColor;
FragColor = vec4(result, 1.0);
```



Light Components / Specular



http://hucoco.com/2018/07/09/Lighting_and_Materials/

Abstraction

A light is an entity in your 3D scene with a position and direction (not all lights have direction).
Material is defining how is incoming light transformed into the final visible color of an object.

Material

Material is a set of properties of an object that define how the object surface affects the incoming light rays.

Predefined materials include shaders that use the following standard lighting models = calculating the final color of a pixel based on all light components.

- Flat
- Phong
- Blinn-Phong
- Lambert
- **Physically based rendering (PBR)**

Standard Shaders / Flat

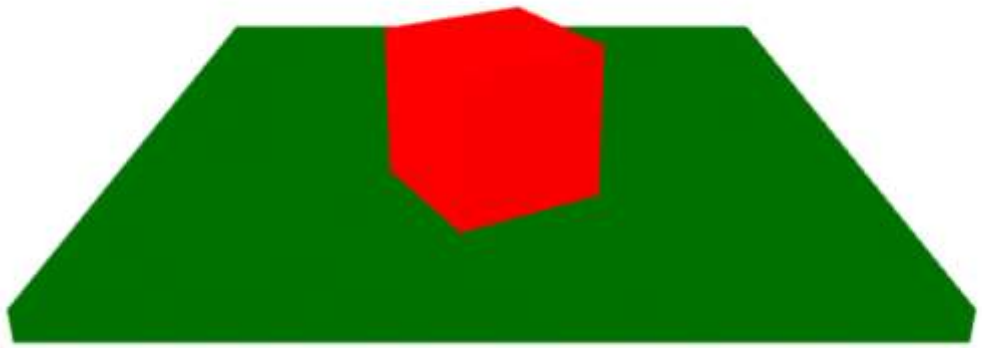
Not affected by any lights

All colors rendered at 100% intensity, no shades or reflections

No computations, super fast

Looks very unrealistic

Suitable for UI elements or small objects

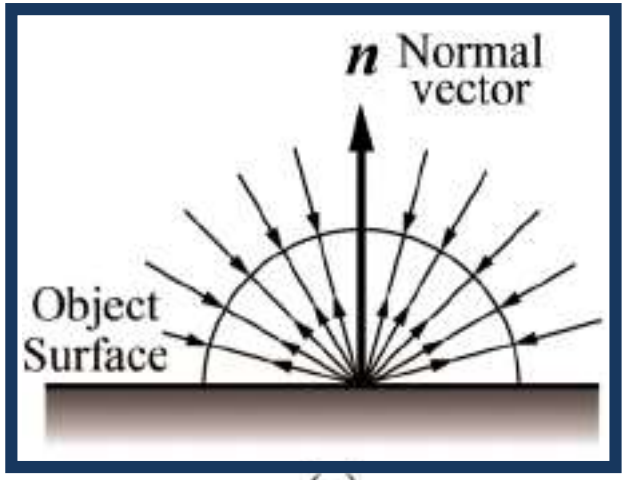


Standard Shaders / Phong

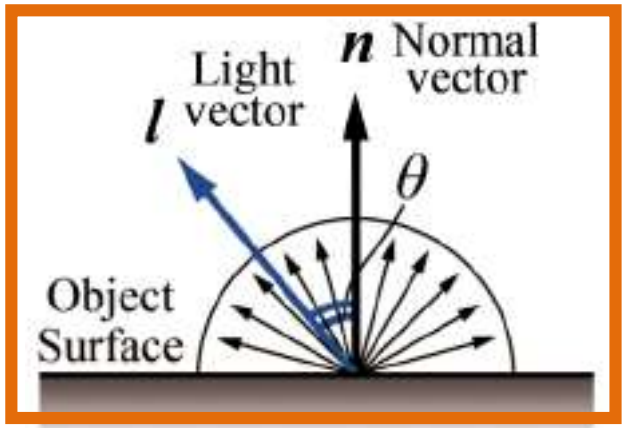
- Requires more fine tuning to achieve realistic look compared to Blinn-Phong or PBR
- Inexpensive and fast
- Ambient + diffuse + specular component makes the final intensity
- Control parameters:

  shininess,

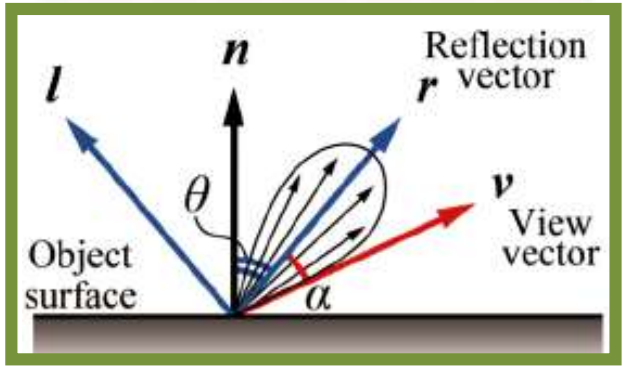
  reflectivity
- A better but less effective alternative to Phong is Blinn-Phong




```
float ambientStrength = 0.1;
vec3 ambient = ambientStrength * lightColor;
```




```
vec3 norm = normalize(Normal);
vec3 lightDir = normalize(lightPos - FragPos);
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * lightColor;
```




```
vec3 viewDir = normalize(viewPos - FragPos);
vec3 reflectDir = reflect(-lightDir, norm);
float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
vec3 specular = specularStrength * spec * lightColor;
```

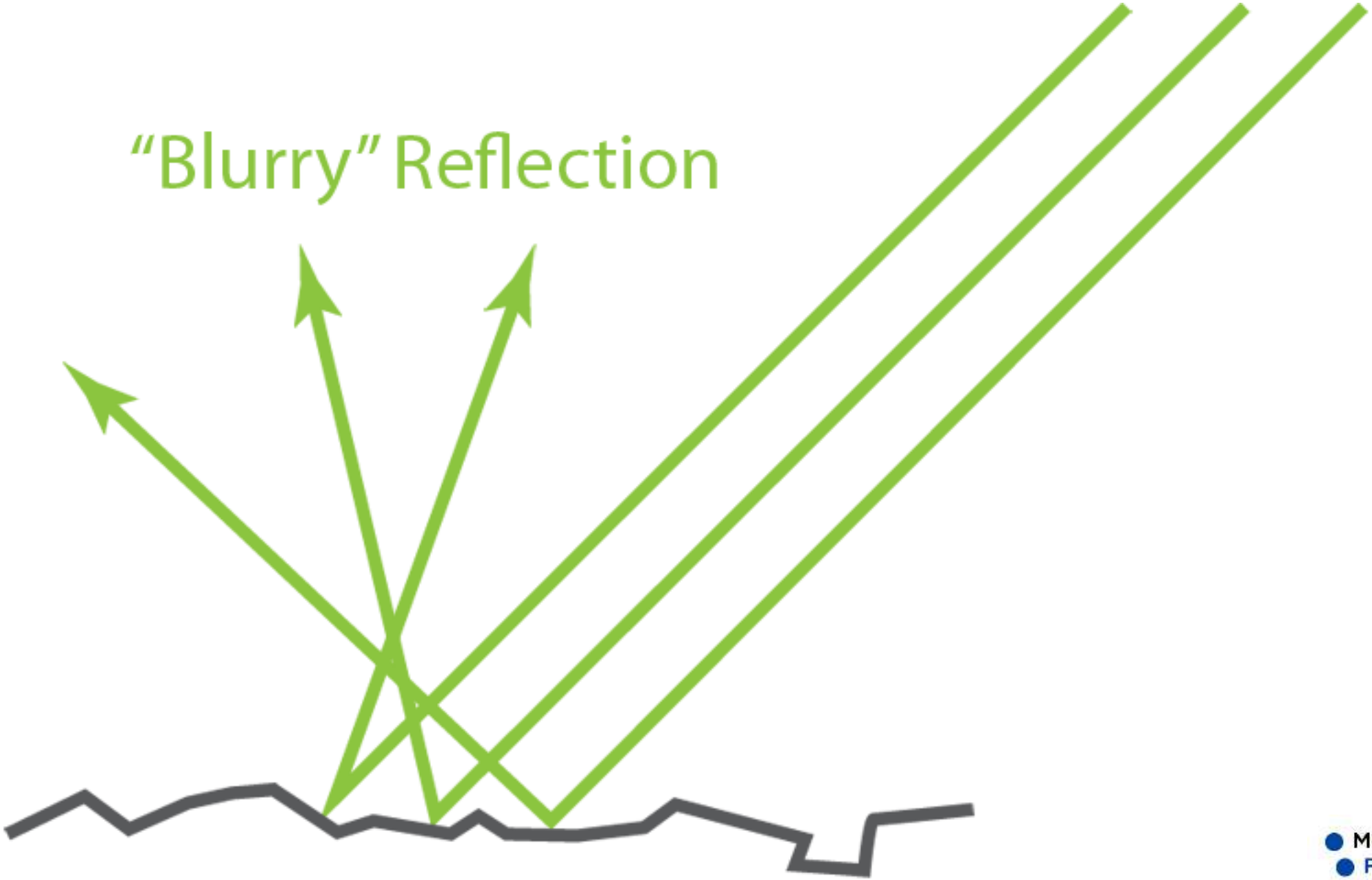
```
vec3 result = (ambient + diffuse + specular) * objectColor;
FragColor = vec4(result, 1.0);
```

Standard Shaders / Physically Based Rendering (PBR)

- Realistic
- Instead of approximations it uses a physically correct model
- Most computationally expensive
- Standard in A-FRAME, Unity, Unreal Engine
- Less configuration than Phong, a lot of is calculated automatically
- Influencing the object appearance using two main parameters:
  - Reflectivity/metalness (odrazivost)
  - Roughness/gloss (lesk)

https://marmoset.co/posts/basic-theory-of-physically-based-rendering/

Disney again: https://media.disneyanimation.com/uploads/production/publication_asset/48/asset/s2012_pbs_disney_brdf_notes_v3.pdf



https://marmoset.co/posts/basic-theory-of-physically-based-rendering/

Increasing Reflectivity
(Constant Albedo)

Increasing Gloss
(Constant Reflectivity)

MENDELU
Faculty
of Business
and Economics

https://marmoset.co/posts/basic-theory-of-physically-based-rendering/

Standard Shaders / Physically Based Rendering (PBR)

- Polished metal, like stainless steel fence:
  - reflectivity: high, gloss: high
- Matte plastic LEGO brick:
  - reflectivity: low, gloss: low
- Vinyl floor:
  - reflectivity: low-moderate, gloss: moderate
- Liquid water:
  - reflectivity: low-moderate, gloss: moderate-high
- Solid ice:
  - reflectivity: moderate-high, gloss: high

Standard Shaders / Physically Based Rendering (PBR)

https://marmoset.co/posts/basic-theory-of-physically-based-rendering/

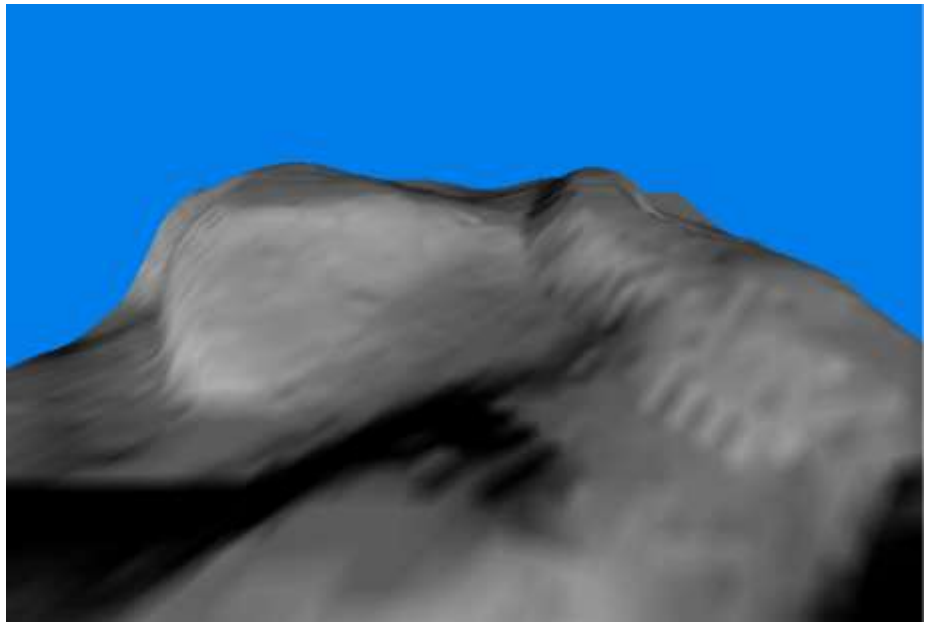
Light

Light is an entity in the scene defining light rays emitted into the scene that are processed by materials of objects.

Ambient
Directional
Hemisphere
Point
Spot
Probe

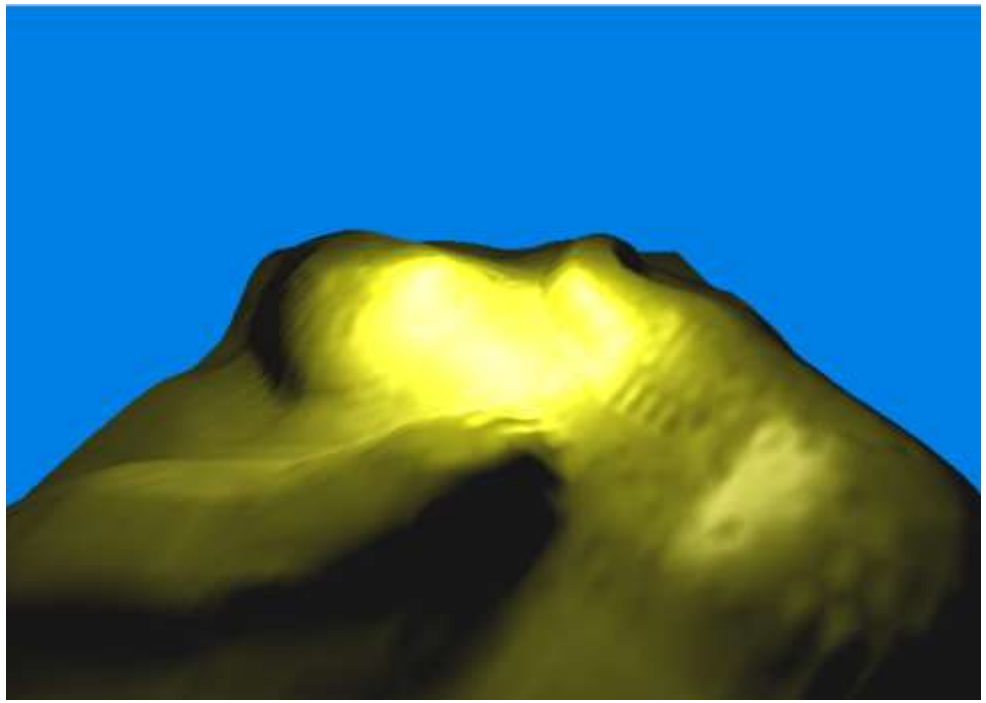
Lights / Directional

- Position is irrelevant
- Always positioned very far away (like [–9999, 9999, 9999])
- Only the direction it targets matter
- Simulates light from a distant light source like Sun
- No decay of light rays on the way = constant ray intensity in the whole scene
- Effective for casting shadows



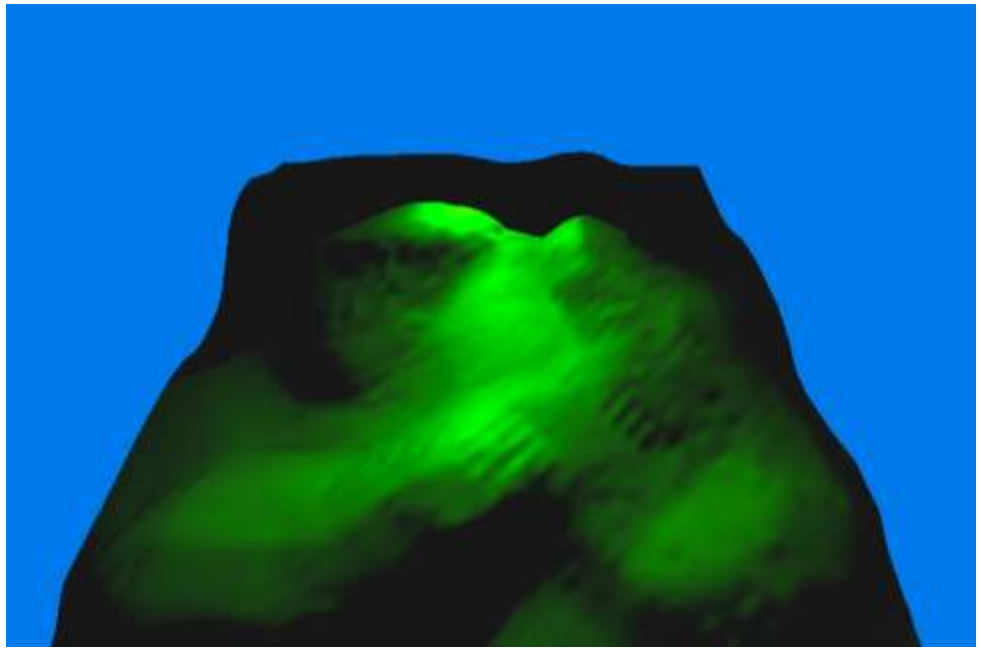
Lights / Point

- Has a specific position
- Has a decay over distance = decreasing intensity further away from the light
- E.g. a bulb or street light



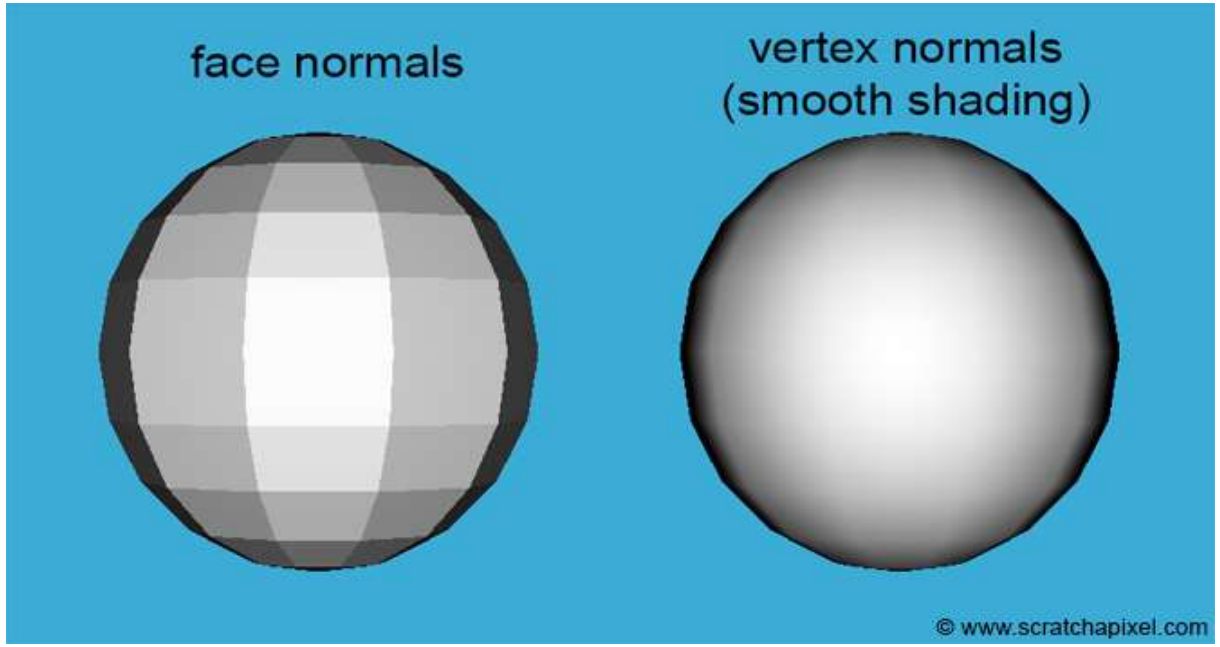
Lights / Spot

- Like the point light with a shield defined by a cutoff angle
- In contrast to point light, spot light shines only in one direction
- Also has a decay over distance
- E.g. a theatre reflector, a torch or a desk lamp

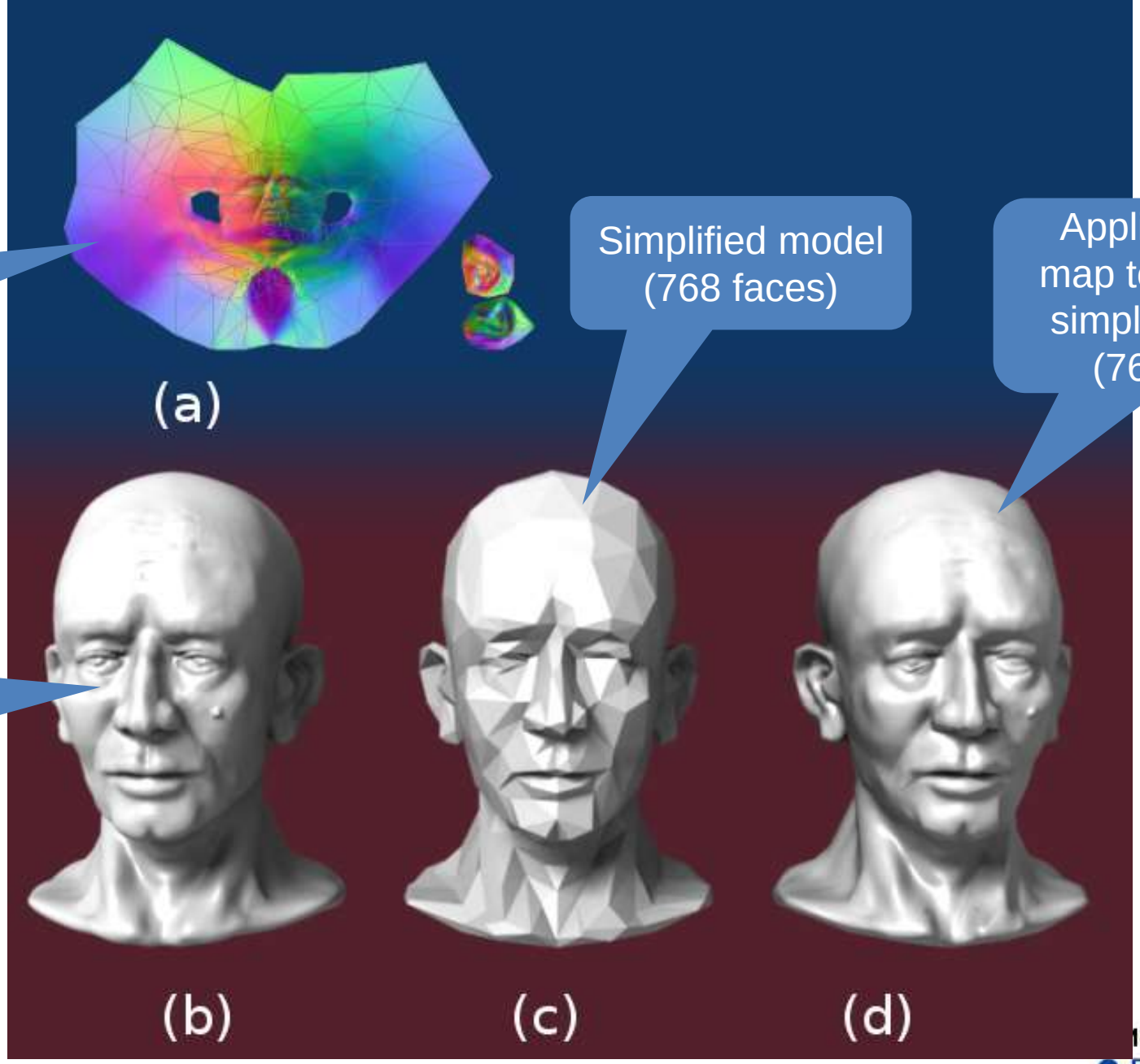


Normals

Normals are vectors attached to vertices of an object
Normal vector is related to the surrounding faces
Therefore, one vertex may have multiple normal vectors
Normals may improve lightning if set correctly because automatic calculations of normal is not always optimal



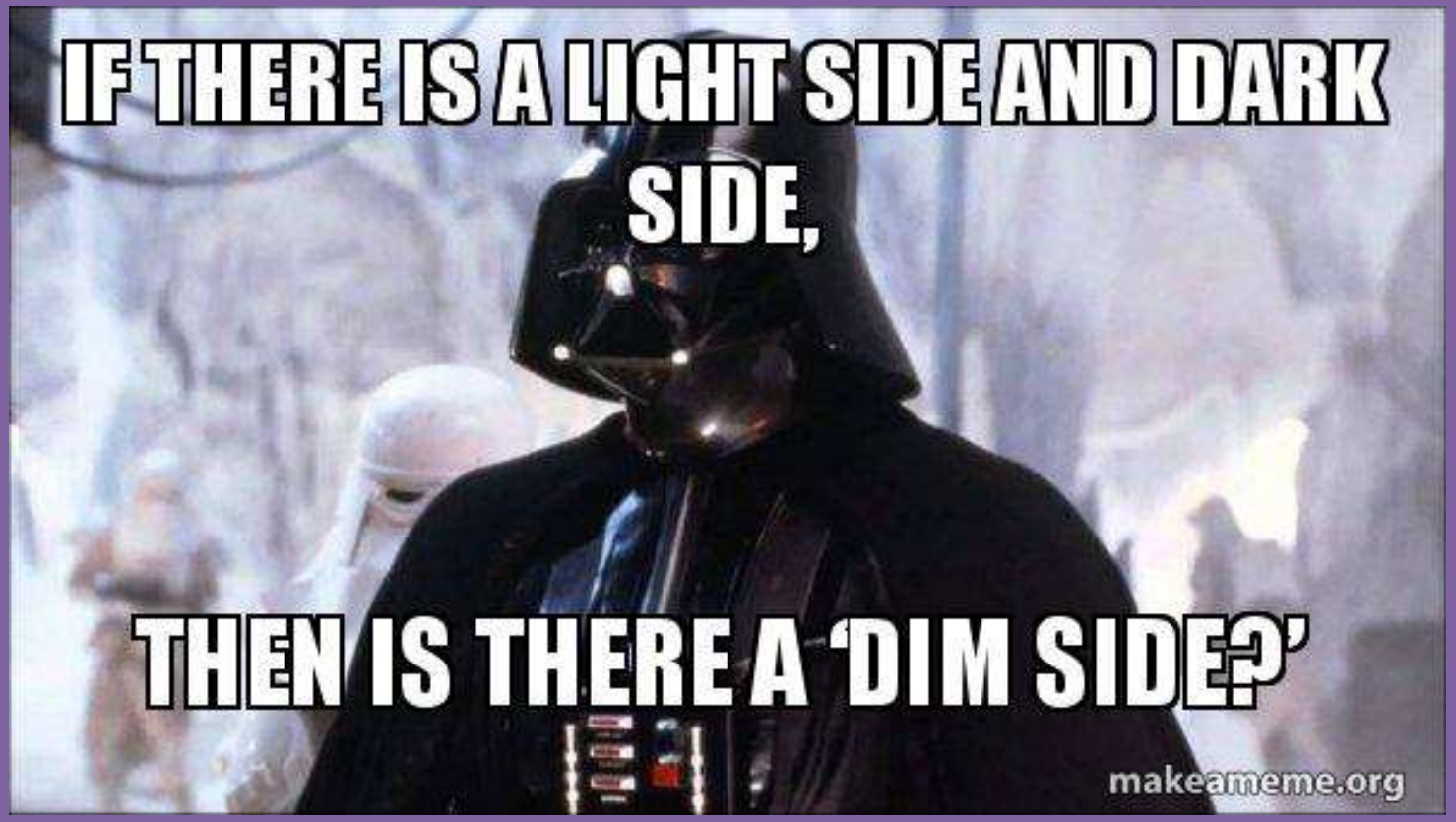
Normal maps



https://en.wikipedia.org/wiki/Normal_mapping

Shadow

1. Light
2. **Shadow**
3. Texture



Shadow

Ambient light does not support shadows.
Light types that support shadows: point, spot, directional.

Shadows are computationally expensive, cannot be turned on for everything similarly to physics.

Possible implementations:

a. “hardcode” shadows into textures and save resources
b. Turn on shadow casting only for selected lights and objects
c. Optimize “shadow camera” to decrease shadow resolution and region of scene to apply shadows in.

May take time to configure realistic and effective shadows.

Texture

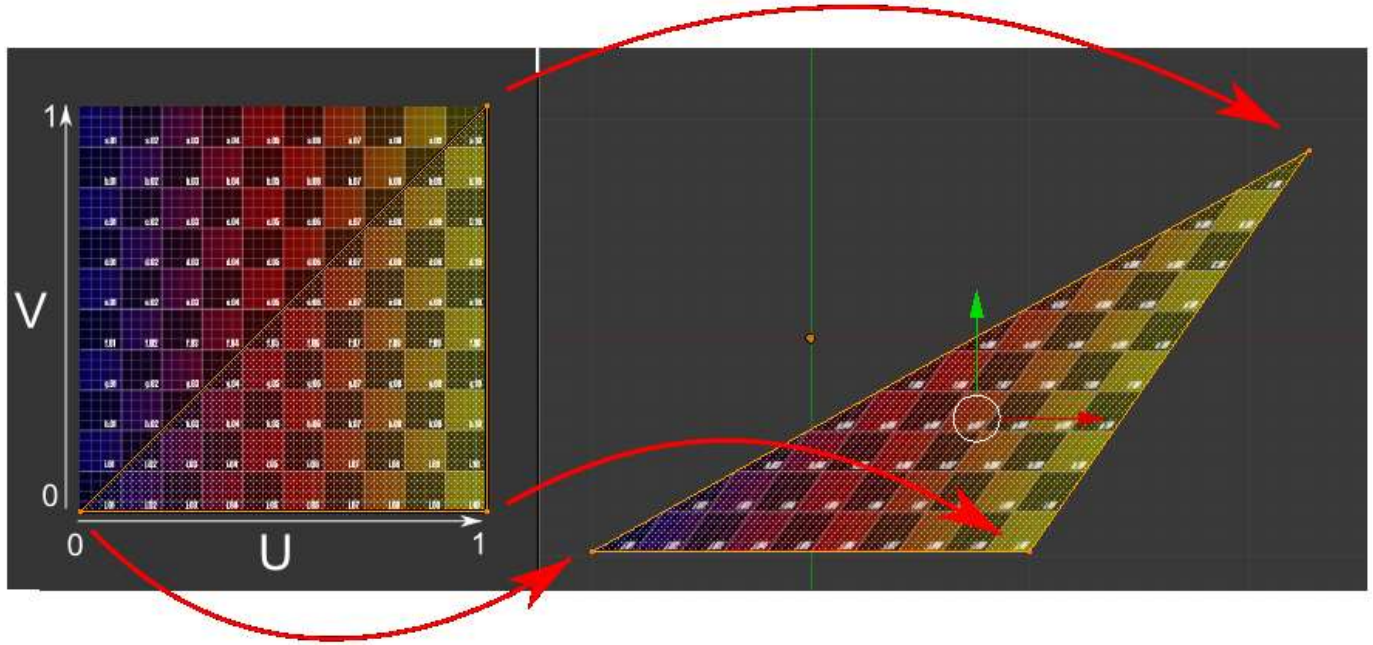
1. Light
2. Shadow
3. **Texture**

Texture

- The texture is matrix of natural numbers for OpenGL API.
- Textures are stored on a graphics card in a similar way the vertices are.
- The data are store in a texture objects that contains the data, description of structure etc.
- The shaders then access the object and read the data.
- A single object may combine different textures – multitexturing.

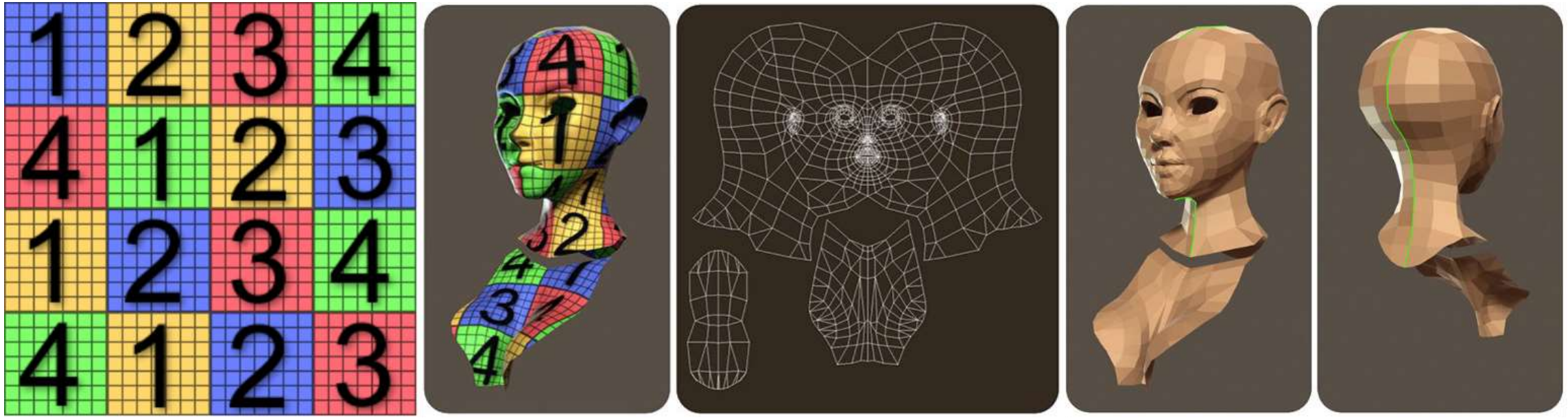
Attaching textures to objects

- 2D texture is attached using UV coordinates on <0, 1> interval.
- Usually, 2D texture has [0, 0] coordinates in bottom left corner and
- [1, 1] in top right corner.
  If we use higher number, the texture repeats (2 = 2×).
- The coordinates are described using letters s, t, r a q.



https://www.opengl-tutorial.org/

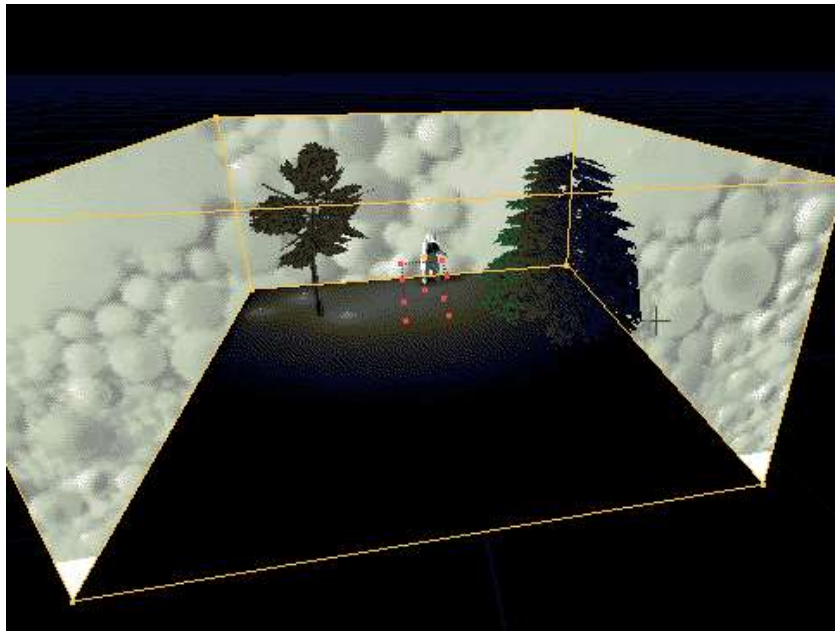
Attaching textures to objects

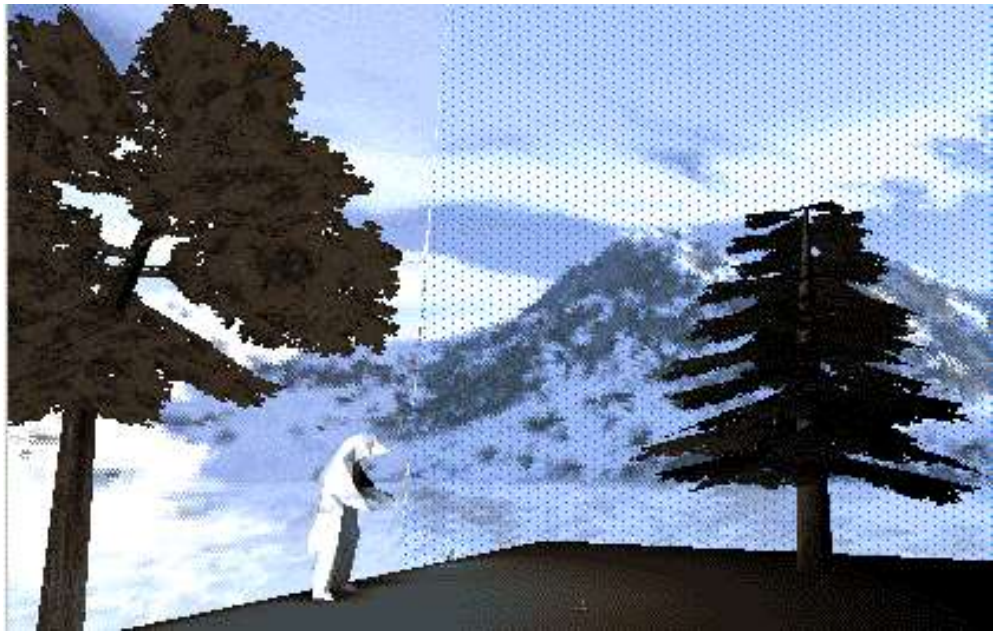


https://cgi.tutsplus.com/articles/ game-character-creation-series-kila-chapter-3-uv-mapping--cg-26754

Skybox Texture

- For many games, we must create an infinite background (e.g. in Doom).
- This background is represented by a cube with textures called Skybox.
- The observer is inside the cube.
- We never apply rotation or translation on this cube!





http://creativegames.org.uk/modules/Game_Design/units/unit10.htm

http://scmapdb.com/skybox:doom1

Environmental Mapping

- *Environmental mapping* = mapping of surrounding environment on rendered object.
- We can simulate (*reflection*) or (*refraction)*.
- There are two possible implementations: sphere map and cube map.
- *Sphere maps* is a technique developed by Blinn and Newell.
- The source image is a spherical projection of surrounding on a single texture.
- It’s not frequently used because of complex preparation.

Environmental Mapping / Sphere map

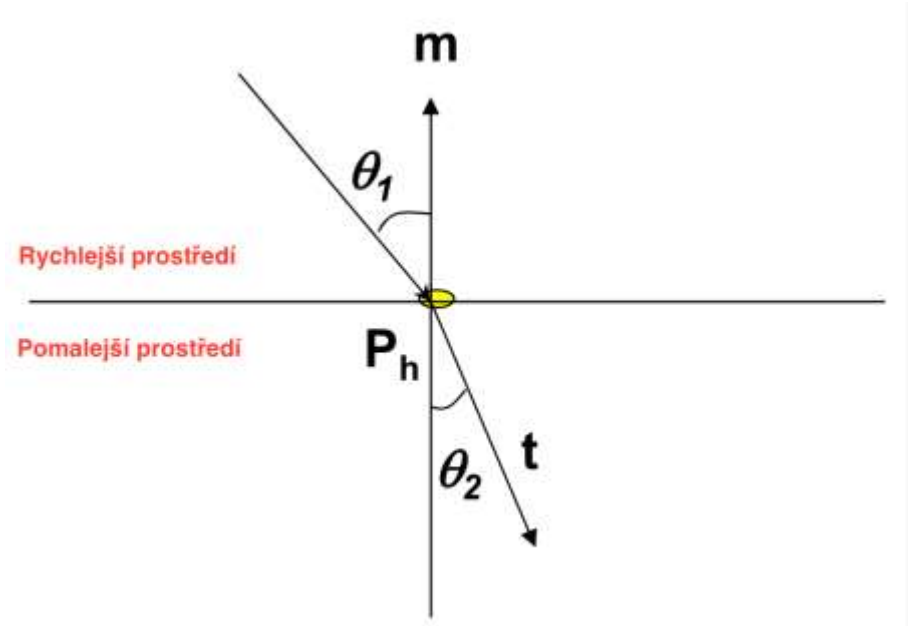
http://www.debevec.org

Environmental Mapping / Refraction and reflection of light

http://www.nvidia.com

Refraction

- Transition from faster medium to slower medium: vector goes to the normal.
- Transition from slower medium to faster medium: vector goes from the normal.
- The *c*1 and *c*2 ratio is important. Usually it is calculated like vacuum/medium ratio.
  - Air: 99.97 %
  - Glass: 52.2 % to 59 %
  - Water: 75.19 %
  - Sapphire: 56.50 %
  - Diamond: 41.33 %

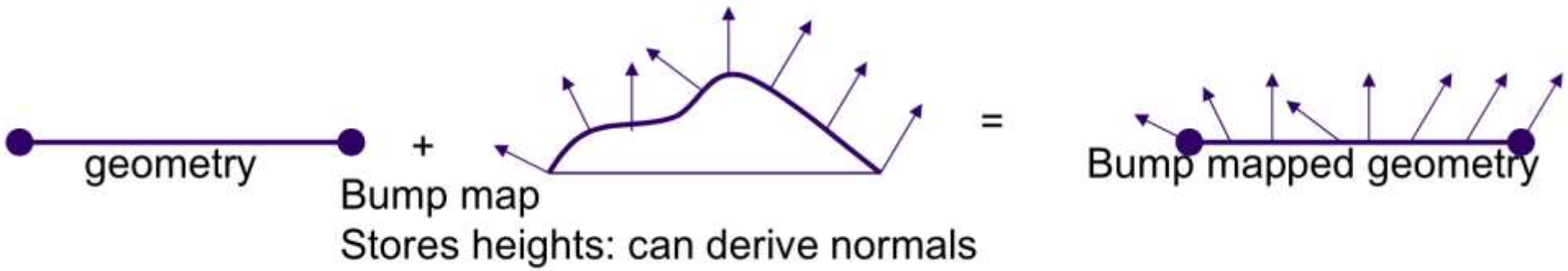


Advanced Textures

Reflection mapping

Advanced Textures

Bump mapping



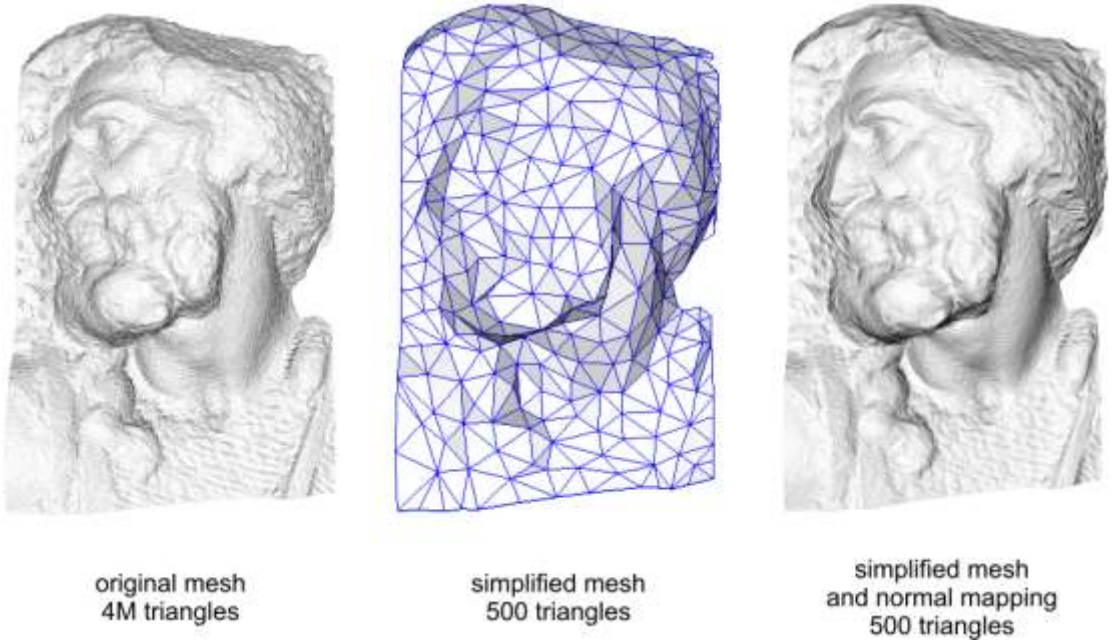
- Invented by Blinn.
- We modify the normal using a value in a texture.
- Further, we calculate lighting using this normal.
- Much more efficient than store the information in the geometry.

Advanced Textures

Bump mapping

Advanced Textures

Normal mapping



Similar to bump mapping, but *bump map* can be applied on different objects. Normal map is for particular object – it contains values of normals.

Wikipedie

Advanced Textures

Alpha & transparency

So called *blending* allows to create translucent objects.
We can also set alpha channel in fragment shader:

```
gl_FragColor = vec4(color.r, color.g, color.b, 0.5)
```

Or skip a pixel completely:

```
outColor = texture(texture0, texCoord0.st);
if (outColor.a < 0.1) {
   discard;
} else {
   gl_FragColor = outColor
}
```


COME TO THE DARK SIDE
WE HAVE RESPIRATOR MASKS
imgflip.com

Reference

https://learnopengl.com/Lighting/Basic-Lighting
https://marmoset.co/posts/basic-theory-of-physically-based-rendering/