Augmented Reality im Web

Workshop von Methusshan Elankumaran

Wie schätzt ihr eure Kenntnisse im Bereich 3D-Rendering ein?

Zeitplan

- Crashkurs: 3D-Rendering mit three.js ~ 50 Min
  - Szene und Kamera erstellen ~ 5 Min
  - Aufbau eines 3D-Modells & Buffering ~ 5 Min
  - Materials ~ 5 Min
  - Texturen ~ 5 Min
  - Transformationen ~ 5 Min
  - Licht ~ 5 Min
  - Aufgabe 1 ~ 20 Min
- Pause ~ 5 - 10 Min
- Augmented Reality mit three.js ~ 45 Min
  - Was ist Augmented Reality? ~ 5 Min
  - Aufbau einer AR-Szene in three.js ~ 10 Min
  - Hit-testing ~ 10 Min
  - Aufgabe 2 ~ 20 Min

Crashkurs:
3D-Rendering mit three.js

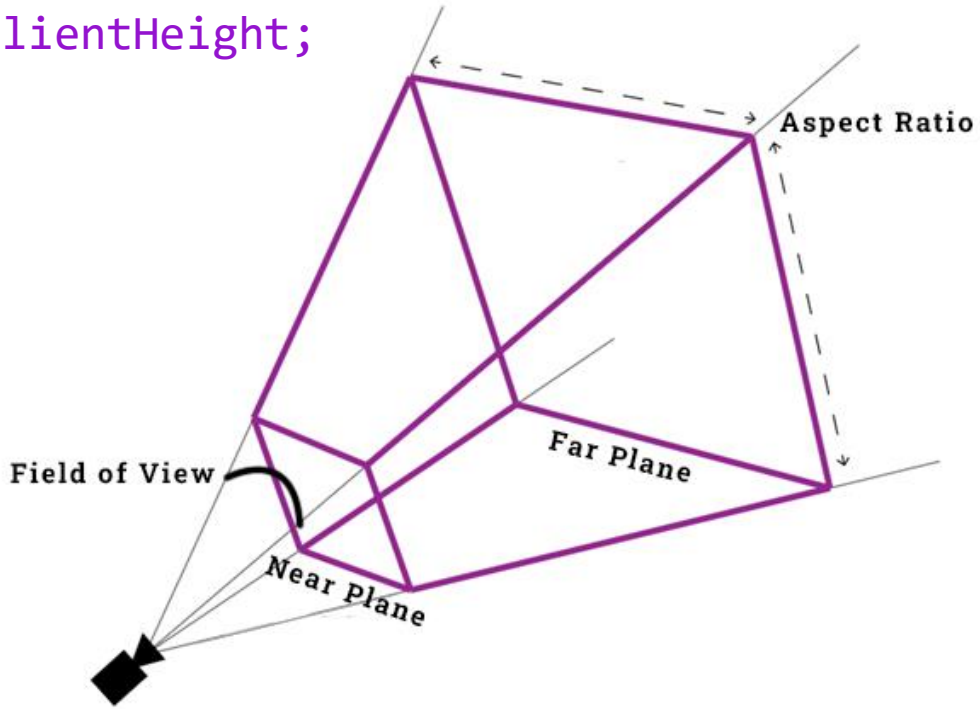
Szene erstellen

- In einer Szene werden alle Objekte einer 3D-Szene gespeichert

```
const scene = new THREE.Scene()
```

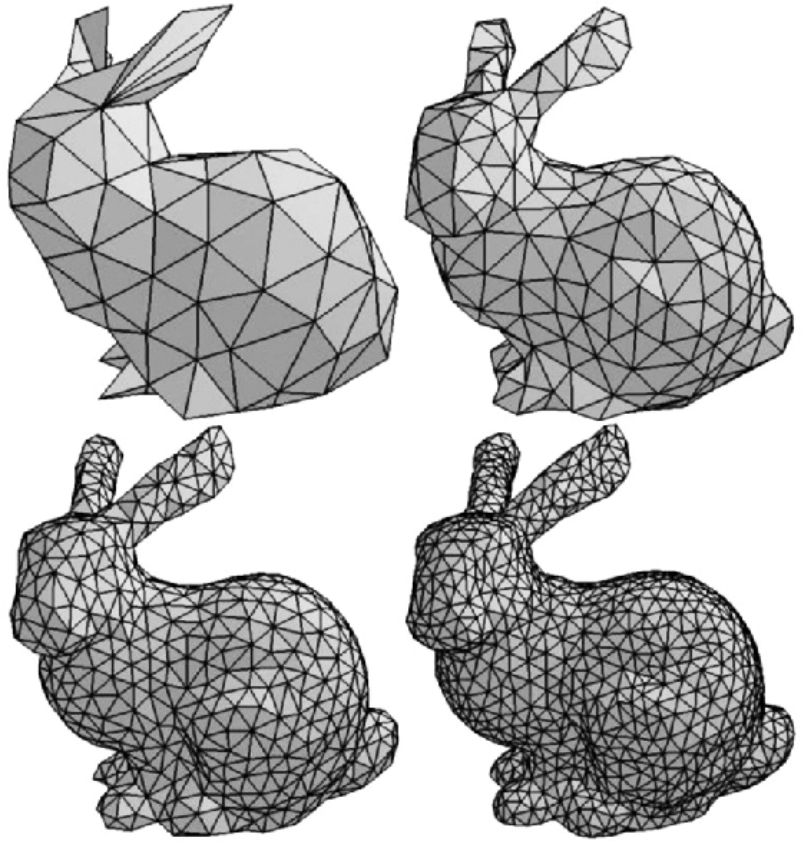
Kamera erstellen

```
const fieldOfView = 55;
const aspectRatio = canvas.clientWidth / canvas.clientHeight;
const nearPlane = 0.1;
const farPlane = 100;


const camera = new THREE.PerspectiveCamera(
    fieldOfView,
    aspectRatio,
    nearPlane,
    farPlane
);
```



Aufbau eines 3D-Modells

- 3D-Modelle bestehen aus vielen Dreiecken
- Dreieckstruktur dient zur einfachen Speicherung der Daten
- Eckpunkte werden in sogenannte Buffer gespeichert



Geometrie in WebGL

Vertex Buffer Objects (VBO)

- Speicherort von Koordinaten in der GPU
- Verwendung der Daten im Vertex Shader

```
float vertices[] = {
    0.5f, 0.5f, 0.0f, 1.0f, 0.2f, 0.3f,
    0.5f, -0.5f, 0.0f, 1.0f, 0.2f, 0.3f,
    -0.5f, 0.4f, 0.0f, 1.0f, 0.2f, 0.3f
    -0.5f, -0.5f, 0.0f, 0.7f, 0.4f, 0.3f,
    0.5f, -0.5f, 0.0f, 1.0f, 0.2f, 0.3f,
    -0.5f, 0.4f, 0.0f, 1.0f, 0.2f, 0.3f
}
```

Geometrie in WebGL

Vertex Array Objects (VAO)

- Register für VertexAttribPointer
- Weist den Werten Attribute zu

```
float vertices[] = {
    0.5f, 0.5f, 0.0f, 1.0f, 0.2f, 0.3f,
    0.5f, -0.5f, 0.0f, 1.0f, 0.2f, 0.3f,
    -0.5f, 0.4f, 0.0f, 1.0f, 0.2f, 0.3f
    -0.5f, -0.5f, 0.0f, 0.7f, 0.4f, 0.3f,
    0.5f, -0.5f, 0.0f, 1.0f, 0.2f, 0.3f,
    -0.5f, 0.4f, 0.0f, 1.0f, 0.2f, 0.3f
        (position)          (color)
}
```

Bufferobjekte in WebGL

Index Buffer Objects (IBO)

- Weist den Knotenpunkten Indexwerte zu
- Verhindert Duplikate

```
float vertices[] = {
    0.5f, 0.5f, 0.0f, 1.0f, 0.2f, 0.3f,
    0.5f, -0.5f, 0.0f, 1.0f, 0.2f, 0.3f,
    -0.5f, 0.4f, 0.0f, 1.0f, 0.2f, 0.3f
    -0.5f, -0.5f, 0.0f, 0.7f, 0.4f, 0.3f,
    0.5f, -0.5f, 0.0f, 0.7f, 0.4f, 0.3f,
    -0.5f, 0.4f, 0.0f, 0.7f, 0.4f, 0.3f
        (position)          (color)
}
```

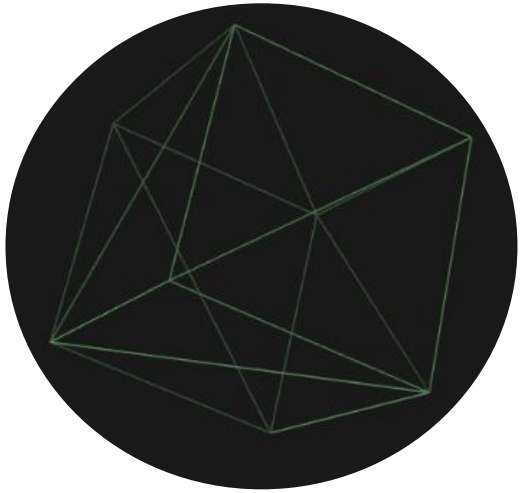
```
float vertices[] = {
   (0) 0.5f, 0.5f, 0.0f, 1.0f, 0.2f, 0.3f,
   (1) 0.5f, -0.5f, 0.0f, 1.0f, 0.2f, 0.3f,
   (2) -0.5f, 0.4f, 0.0f, 1.0f, 0.2f, 0.3f
   (3)-0.5f, -0.5f, 0.0f, 0.7f, 0.4f, 0.3f,
        (position)          (color)
}
```

```
float indices[] = {
    0,1,2
    3,1,2
}
```

Definition der Geometrie in three.js

Würfel/Quader:

```
const cubeSize = 4;
const cubeGeometry = new THREE.BoxGeometry(
        cubeSize,
        cubeSize,
        cubeSize);
```

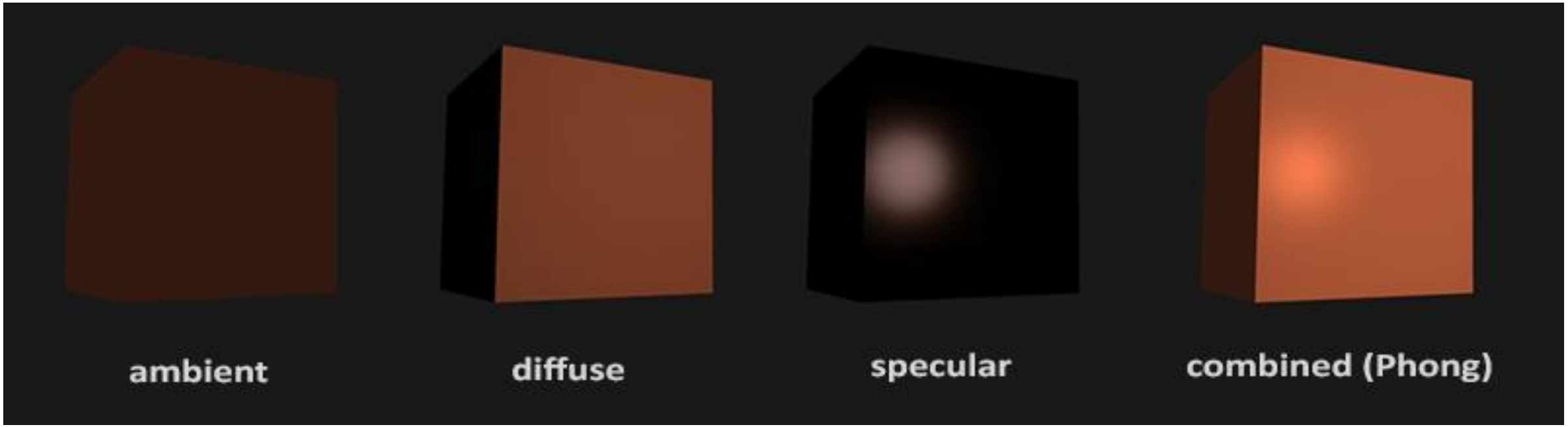


Sphäre:

```
const sphereRadius = 3;
const sphereWidthSegments = 32;
const sphereHeightSegments = 16;
const sphereGeometry = new THREE.SphereGeometry(
        sphereRadius,
        sphereWidthSegments,
        sphereHeightSegments
 );
```

Materials

- Phong-Material
  - Vereinfachtes Lichtmodell
  - 3-Lichtarten: Ambient, Diffuse, Specular



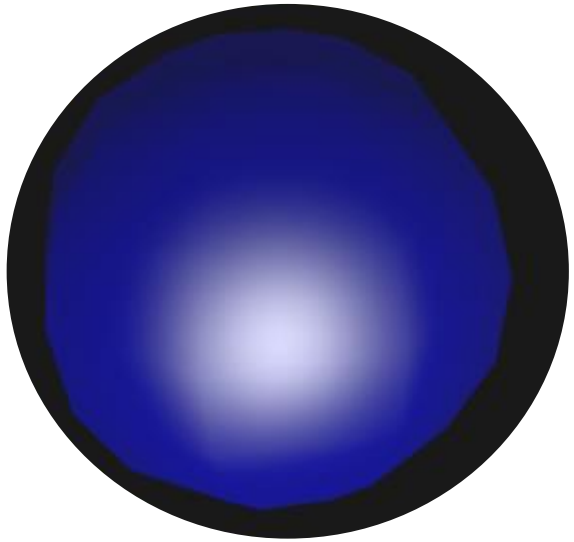
Materials

- Lambert-Material
  - Vereinfachtes Lichtmodell
  - 2-Lichtarten: Diffuse, Ambient

Materials

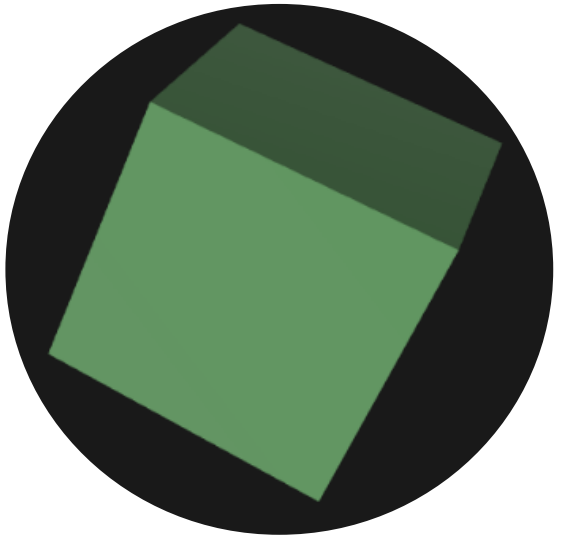
- Phong:

```
const sphereMaterial =
new THREE.MeshPhongMaterial({
        color: 'blue',
        specular: 'white',
        shininess: 20
    })
```



- Lambert:

```
const cubeMaterial =
new THREE.MeshLambertMaterial({
        color: 'lightgreen'
    });
```



Texturen – Texture Wrapping

Repeat

Mirrored Repeat

Clamp to Edge

Clamp to Border

Jeweils für beide Koordinatenachsen einstellbar

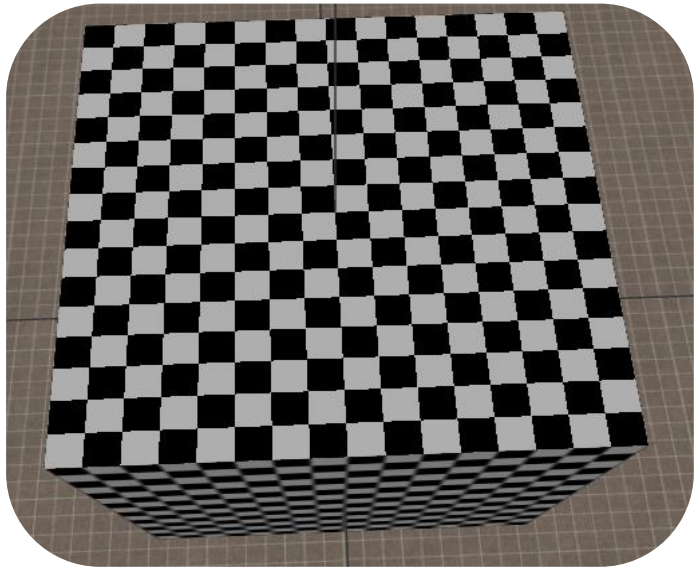
Texturen – Texture Wrapping

```
// Texture Loader erstellen
const textureLoader = new THREE.TextureLoader();

// Textur laden
const planeTextureMap = textureLoader.load('textures/pebbles.jpg‘);

// Texture Wrap für beide Achsen einstellen
planeTextureMap.wrapS = THREE.RepeatWrapping;
planeTextureMap.wrapT = THREE.RepeatWrapping;

// Wiederholung der Textur einstellen
planeTextureMap.repeat.set(16, 16);
```

Texturen – Texture Filtering

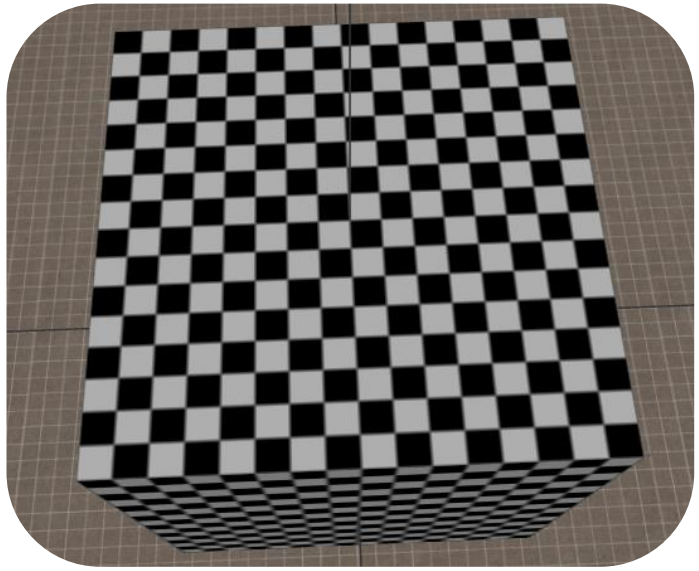
Nearest:

- Gibt immer den Farbwert des nächsten Pixels zurück



Linear:

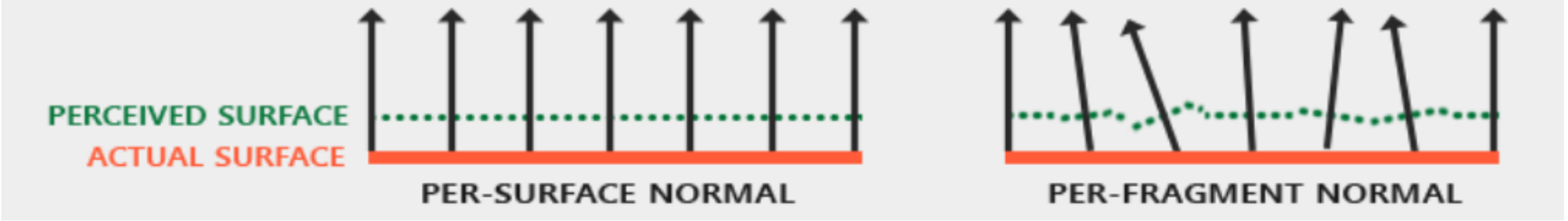
- Interpoliert zwischen den umliegenden Pixeln



Kann eingestellt werden für Vergrößerungs- und Verkleinerungs-Operationen

Texturen – Normal Mapping

- Anpassung der Normalenwerte eines Objektes durch eine Normal Map



Texturen – Normal Mapping

Ohne Normal Map

Mit Normal Map

Texturen – Einfügen von Texture und Normal Map in ein Material

```
const planeMaterial = new THREE.MeshLambertMaterial({
        map: planeTextureMap,
        side: THREE.DoubleSide,
        normalMap: planeNorm
    })
```

Geometrie und Material/Texturen zusammenfügen

```
//Mesh definieren
const cube = new THREE.Mesh(cubeGeometry, cubeMaterial);
Position festlegen
cube.position.set(cubeSize + 1, cubeSize + 1, 0);
// Mesh zur Szene Hinzufügen
scene.add(cube);
```

Transformationen

Translation:

$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + T_x \\ y + T_y \\ z + T_z \\ 1 \end{pmatrix}$$

Homogene Koordinaten

Translation:

$$\begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & -2 \\ 0 & 0 & 2 \end{bmatrix} * \begin{pmatrix} 2 \\ 2 \\ 2 \end{pmatrix} = \begin{pmatrix} 6 \\ -2 \\ 4 \end{pmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{pmatrix} 2 \\ 2 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 0 \\ 4 \\ 1 \end{pmatrix}$$

Transformationen

Skalierung:

$$\begin{bmatrix} S_1 & 0 & 0 & 0 \\ 0 & S_2 & 0 & 0 \\ 0 & 0 & S_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} S_1 * x \\ S_2 * y \\ S_3 * z \\ 1 \end{pmatrix}$$

Transformationen

Rotation um Achsen:

Rotation um die X-Achse

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ \cos\theta * y - \sin\theta * z \\ \sin\theta * y + \cos\theta * z \\ 1 \end{pmatrix}$$

Rotation um die Y-Achse

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta * x + \sin\theta * z \\ y \\ -\sin\theta * x + \cos\theta * z \\ 1 \end{pmatrix}$$

Rotation um die Z-Achse

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} * \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta * x - \sin\theta * y \\ \sin\theta * x + \cos\theta * y \\ z \\ 1 \end{pmatrix}$$

Transformationen

Transformationen in Three.js

Translation:

```
(Mesh).position.set(x,y,z) oder (Mesh).position.x/y/z = Wert
```

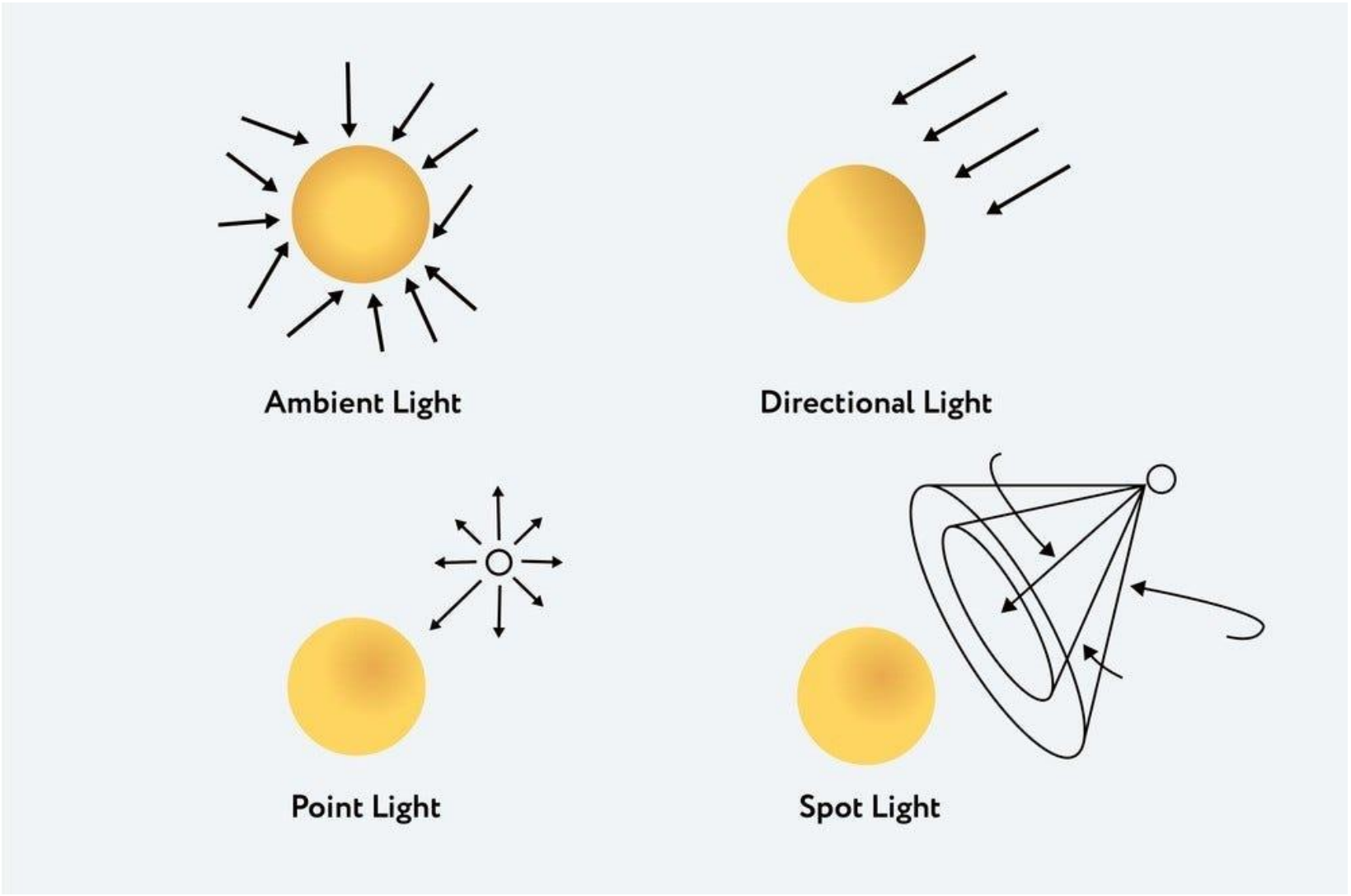
Skalierung:

```
(Mesh).scale.set(x,y,z) oder (Mesh).scale.x/y/z = Wert
```

Rotation:

```
(Mesh).rotation.set(x,y,z) oder (Mesh).rotation.x/y/z = Wert
```

Licht



Licht

Ambient Light:

```
const ambientColor = 0xffffff;
const ambientIntensity = 0.2;
const ambientLight = new
THREE.AmbientLight(ambientColor,
ambientIntensity);
scene.add(ambientLight);
```

Point Light:

```
const pointColor = 0xff0000;
const pointIntensity = 1;
const distance = 100;
const distanceLight = new
THREE.PointLight( pointColor,
pointIntensity, pointDistance);
distanceLight.position.set( 50, 50, 50
);
scene.add( distanceLight );
```

Directional Light:

```
const directionalColor = 0xffffff;
const directionalIntensity = 1;
const directionalLight = new
THREE.DirectionalLight(directionalColo
r, directionalIntensity);
directionalLight.position.set( 50, 50,
50 );

directionalLight.target = (Mesh);
scene.add(directionalLight);
```

Animation

```
function animate(time) {
    time *= 0.001
    requestAnimationFrame(animate)
    // Animationseinstellungen bspw. sphere.rotation.x += 0.005;
    render()
}
animate()

function render() {
    gl.render(scene, camera)
}
```

Aufgabe 1

Erstelle eine Szene in three.js, die eine Kugel beinhaltet. Diese soll die zwei Texturen im texture-Folder beinhalten. Lasse diese sich kontinuierlich um die X- und Y-Achse drehen. Füge ein Directional Light hinzu, welches auf die Kugel zeigt. Wrapping-Einstellungen für die Texturen können selbst gewählt werden.

Pause

Augmented Reality mit Three.js

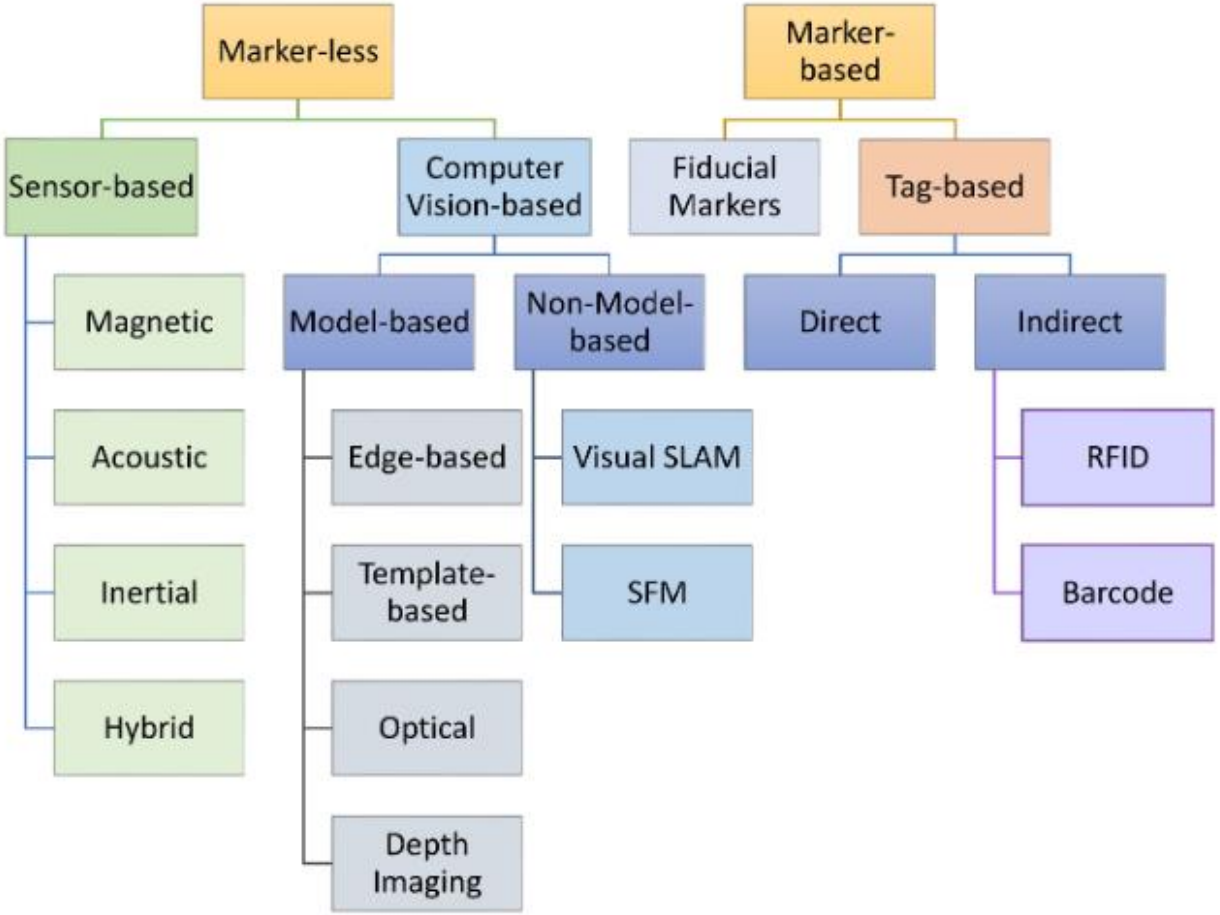
Was ist Augmented Reality?

- Die Erweiterung der Realität durch die Anreicherung mit computergerierten Informationen
- Wird durch eine Kombination aus Hardware- und Software realisiert
- Bspw: Smartphone oder AR-Brillen

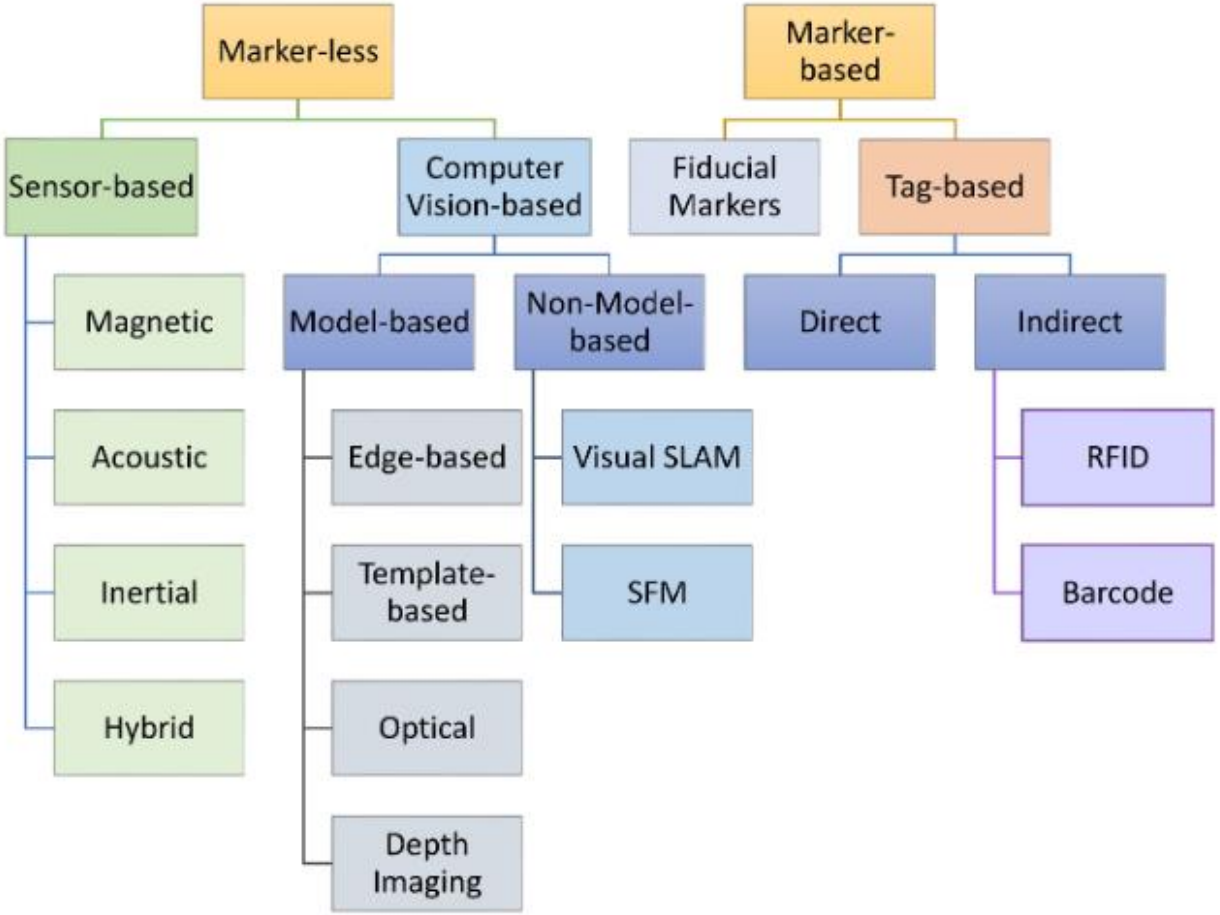
Historie von Augmented Reality

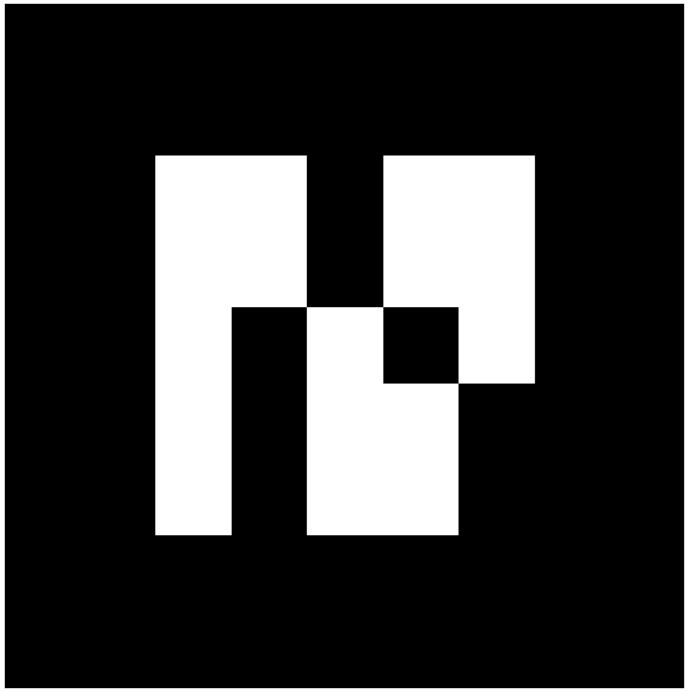
- 1968: Erfindung von AR durch Ivan Sutherland
- 1974: Myron Krüger baut ein HMD, welches auch
  Schatten visualisiert
- 1990: Der Name „Augmented Reality“ wird geboren

Tracking Verfahren



Aufbau einer AR-Szene in Three.js





Aufbau einer AR-Szene in Three.js

- loadScene()
- onRequestSession()
- onSessionStarted()
- setupWebGLLayer()
- animate()
- render(time, frame)
- endXrSession()
- onSessionEnd()

Laden einer Szene

- Laden des Canvas und des WebGL-Layers

```
function loadScene() {
	// setup WebGL
	glCanvas = document.createElement('canvas');
	gl = glCanvas.getContext('webgl', { antialias: true });
	...
}
```

Laden einer Szene

- Erstellen einer Kamera und einer Szene

```
camera = new THREE.PerspectiveCamera(
      70,
      window.innerWidth / window.innerHeight,
      0.01,
      1000
);

scene = new THREE.Scene();
```

Laden einer Szene

- Erstellen eines Lichts

```
var light = new THREE.HemisphereLight( 0xffffff, 0xbbbbff, 1 );
light.position.set( 0.5, 1, 0.25 );
scene.add( light );
```

Reticle erstellen

```
reticle = new THREE.Mesh(
      new THREE.RingBufferGeometry(0.15, 0.2,
      32).rotateX(-Math.PI / 2),
      new THREE.MeshBasicMaterial({color: "#00FF00"})
 );
reticle.matrixAutoUpdate = false;
reticle.visible = false;
scene.add(reticle);
```

Laden einer Szene

- Erstellen eines WebGL-Renderers

```
renderer = new THREE.WebGLRenderer({
      canvas: glCanvas,
      context: gl
});
renderer.setPixelRatio( window.devicePixelRatio );
renderer.setSize( window.innerWidth, window.innerHeight );
renderer.xr.enabled = true;
document.body.appendChild( renderer.domElement );
```

Check für Support von WebXR

```
navigator.xr.isSessionSupported('immersive-ar‘)
        .then((supported) => {
                if (supported) {
                        btn = document.createElement("button");
                        btn.addEventListener('click', onRequestSession);
                        btn.innerHTML = "Enter XR";
                        var header = document.querySelector("header");
                        header.appendChild(btn);
                }
```

Check für Support von WebXR

```
        else {
                // create fallback session
                navigator.xr.isSessionSupported('inline‘)
                        .then((supported) => {
                                if (supported) {
                                        console.log('inline session supported');
                                }
                                else {
                                        console.log('inline not supported')};
                                })
         }
})
.catch((reason) => {
        console.log('WebXR not supported: ' + reason);
})
```

XR-Session anfragen

```
function onRequestSession(){
	console.log("requesting session");
	navigator.xr.requestSession(
		'immersive-ar‘,
		{requiredFeatures: [‘hit-test’],
		optionalFeatures: [‘local-floor’]})
			.then(onSessionStarted)
			.catch((reason) => {
				console.log('request disabled: ' + reason.log);
			});
}
```

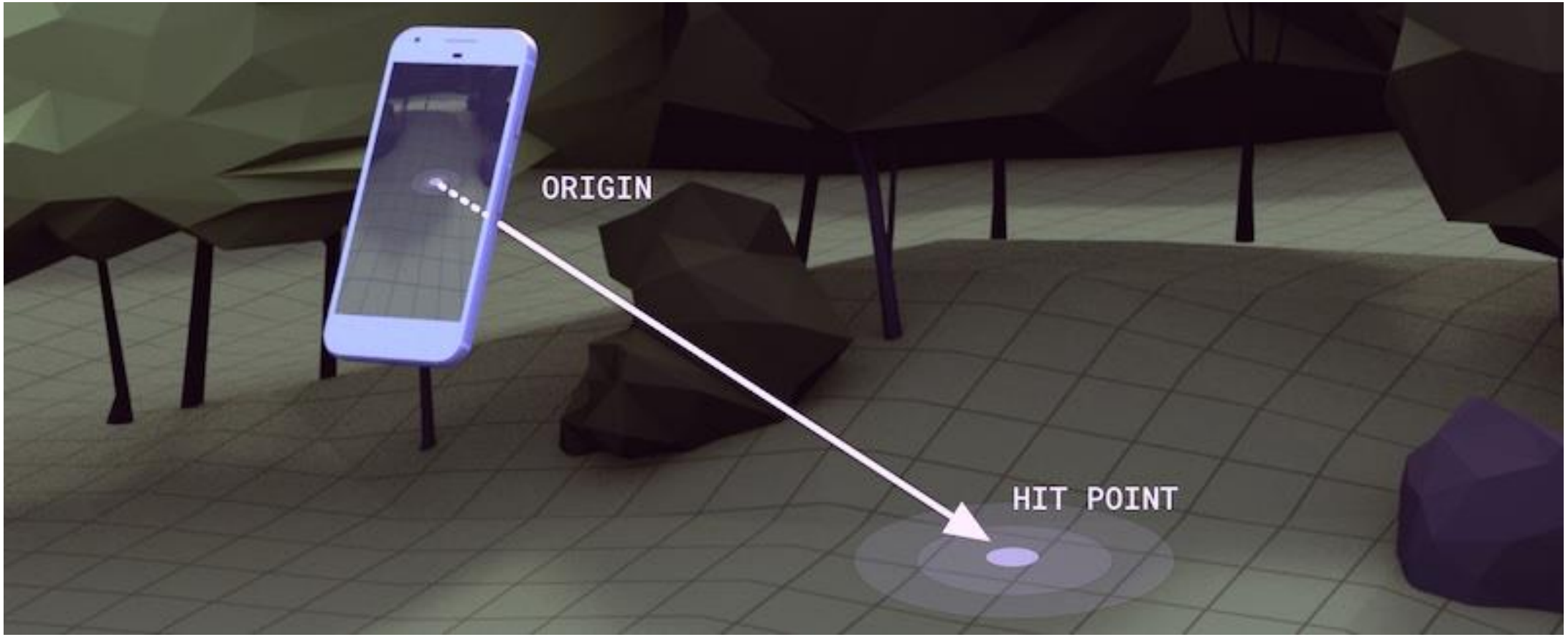
XR-Session starten

```
function onSessionStarted(session) {
    console.log('starting session');
    btn.removeEventListener('click', onRequestSession);
    btn.addEventListener('click', endXRSession);
    btn.innerHTML = "STOP AR";
    xrSession = session;
    xrSession.addEventListener('select', onSelect);
    setupWebGLLayer()
        .then(() => {
            renderer.xr.setReferenceSpaceType('local');
            renderer.xr.setSession(xrSession);
            animate();
        })
}
```

XR-Session starten

```
function setupWebGLLayer() {
    return gl.makeXRCompatible().then(() => {
        xrSession.updateRenderState(
            { baseLayer: new XRWebGLLayer(xrSession, gl) });
    });
}
```

Hit-Testing



Render

```
function render(time, frame) {
	if (frame) {
		var referenceSpace = renderer.xr.getReferenceSpace('local');
		var session = frame.session;
		xrViewerPose = frame.getViewerPose(referenceSpace);
		if (hitTestSourceRequested === false) {
			session.requestReferenceSpace("viewer").then((referenceSpace) => {
			session.requestHitTestSource({space: referenceSpace})
			.then((source) => {hitTestSource = source;})
		});
		session.addEventListener("end", () => { hitTestSourceRequested = false;
		hitTestSource =	null});
	}
```

Render

```
        if (hitTestSource) {
                var hitTestResults = frame.getHitTestResults(hitTestSource);
                if (hitTestResults.length > 0) {
                var hit = hitTestResults[0];
                reticle.visible = true;

        reticle.matrix.fromArray(hit.getPose(referenceSpace).transform.matrix);
        } else {
                reticle.visible = false;
        }
          }
          renderer.render(scene, camera);
}
```

XR-Session beenden

```
function endXRSession() {
     if (xrSession) {
          console.log('ending session...');
          xrSession.end().then(onSessionEnd);
     }
}
```

Zurücksetzen aller Einstellungen

```
function onSessionEnd() {
    xrSession = null;
    console.log('session ended');
    btn.innerHTML = "START AR";
    btn.removeEventListener('click', endXRSession);
    btn.addEventListener('click', onRequestSession);
}
```

Aufgabe 2

Implementiere eine Anwendung, die eine beliebige Anzahl an Vasen auf ebenen Flächen platziert. Es muss nur der Hit-Test implementiert werden, der EventListener zum Laden und Platzieren der Objekte existiert bereits.

Vielen Dank für eure Aufmerksamkeit!