Spacetacular

Grafische Datenverarbeitung SS2015

Stefanie Zahn

Gliederung

1. Demo (Video)
2. Impressionen (Bilder)
3. Systemarchitektur (Abläufe & Objekte)
4. Zeitplan vs. Realität
5. Aufgabenverteilung/Aufwand

Demo (Video)

Spacetacular - ein Weltraumshooter der nächsten Generation

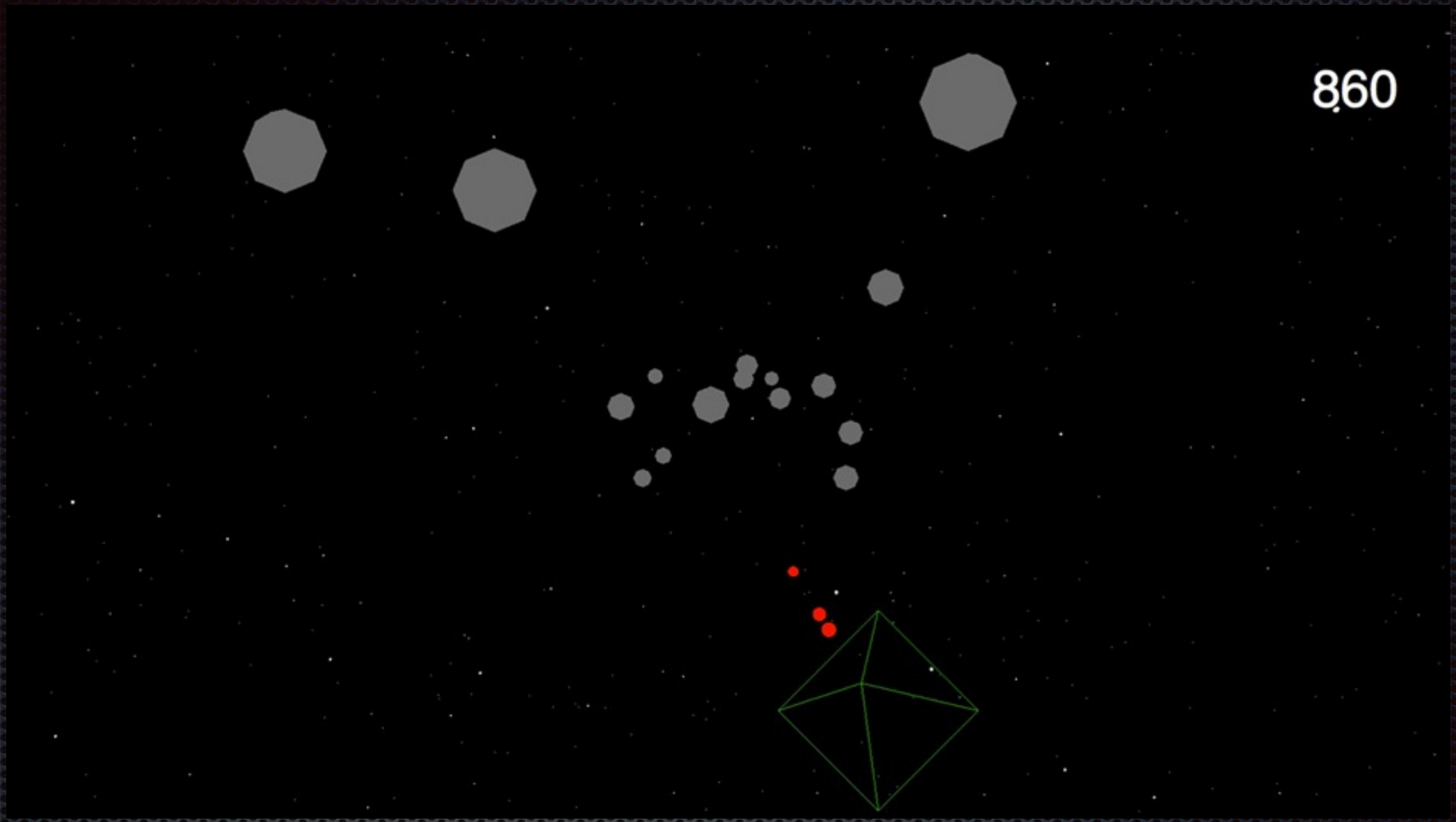
Youtube-Link: https://youtu.be/9zjFusv_jjk

Impressionen

Bilder

Start-Screen (Ausschnitt)

Start-Screen



Spiel-Modus

Game-Over Screen

Systemarchitektur Objekte

- Raumschiff
- Meteoriten
- Sternenhimmel

Systemarchitektur Objekte Raumschiff

```
var shipgeo = new THREE.CylinderGeometry(0, 10, 10, 4, 1, true);
var shipmat = new THREE.MeshBasicMaterial({color: 0x00ff00, wireframe: true});
var ship = new THREE.Mesh(shipgeo, shipmat);

ship.position.y = -20;
ship.position.z = 0;
ship.rotation.x = -0.5*Math.PI;

scene.add(ship);
```

- *Erstellen und hinzufügen des Raumschiffs in die Szene*
- *Wird um 90 Grad nach vorne gedreht*

Systemarchitektur Objekte Meteoriten

```
var meteorits  = [];

for ( i = 0; i < 20; i++) {
  var geometry = new THREE.SphereGeometry(8, 4, 4);
  var material = new THREE.MeshBasicMaterial({color: 0x6a6a6a, wireframe: false});
  var met = new THREE.Mesh(geometry, material);

  met.position.z = (Math.random() * (-1000 - (1000)) + (1000));
  met.position.x = (Math.random() * (85 - (-85)) + (-85));
  met.position.y = (Math.random() * (85 - (-85)) + (-85));

  scene.add(met);
  meteorits.push(met);
}
```

- *Erstellen und hinzufügen der Meteoriten in die Szene*
- *Zufällige Anordnung der Meteoriten*
- *Werden in einem Array gespeichert*

Systemarchitektur Objekte Sternenhimmel

```
var sphere = new THREE.SphereGeometry( 1, 8, 8 );
var starmat = new THREE.MeshBasicMaterial( { color: 0xffffff, specular: 0xffffff, shininess: 50 } );
var stars = [];

for (var i = 0; i < 5000; i ++) {

  var star = new THREE.Mesh(sphere, starmat);

  star.position.x = 3000 * (2.0 * Math.random() - 1.0);
  star.position.y = 3000 * (2.0 * Math.random() - 1.0);
  star.position.z = (Math.random() * (100 - (-3000)) + (-3000));

  scene.add(star);
  stars.push(star);
}
```

- *Erstellen und hinzufügen des Sternenhimmels in die Szene*
- *Zufällige Anordnung der Sterne*
- *Werden in einem Array gespeichert*

Systemarchitektur Funktionen

- Raumschiffsteuerung
- Meteoritenbewegung
- Sternenhimmelbewegung
- Schussfunktion (Projektile)
- Kollision
- Punktezähler

Systemarchitektur Funktionen Raumschiffsteuerung

```
function shipcontrols(event) {
  switch(event.keyCode) {
    //Pfeiltaste oben
    case 38: ship.position.y += 5;
    break;
    //Pfeiltaste unten
    case 40: ship.position.y -= 5;
    break;
    //Pfeiltaste links
    case 37: ship.position.x -= 5;
    break;
    //Pfeiltaste rechts
    case 39: ship.position.x += 5;
    break;
    // Leertaste
    case 32: shoot();
    break;
  }
}
```

```
//Eventlistener für die Tastatureingaben bei 'keydown'-Events. Diese werden
//an die Schiffssteuerung weitergegeben.
window.addEventListener('keydown', shipcontrols);
```

- *Keyboard-Event (Pfeiltasten und Leertaste) löst Steuerung des Schiffes oder Schießen-Funktion aus*

Systemarchitektur Funktionen Meteoritenbewegung

```
function meteorit() {
  for (i = 0; i < meteorits.length; i++) {
    //Meteorit bewegen um x- und y-Achse; z-Position
    //meteorits[i].rotation.x += 0.01;
    //meteorits[i].rotation.y += 0.01;
    meteorits[i].position.z += 2.5;

  //Meteoriten verschieben wenn hinter Kamera
    if (meteorits[i].position.z >= 100) {
      meteorits[i].position.z = (Math.random() * (200 - (-200)) + (-200)) -1000;
      meteorits[i].position.x = (Math.random() * (100 - (-100)) + (-100));
      meteorits[i].position.y = (Math.random() * (100 - (-100)) + (-100));
      meteorits[i].visible = true;
    }
```

- *Elemente des Meteoriten-Arrays werden in der Szene nach vorne bewegt*
- *Wenn Meteoriten hinter der Kamera sind werden sie wieder ganz nach vorne zufällig positioniert*

Systemarchitektur Funktionen Meteoritenbewegung

```
else if (score >= 2500) {
  meteorits[i].position.z += 2.5;
}

else if (score >= 10000) {
  meteorits[i].position.z += 5;
}

if (!go) {
  //random Color-Picker von http://www.paulirish.com/2009/random-hex-color-code-snippets/
  meteorits[i].material.color.setHex('0x'+(Math.floor(Math.random()*16777215).toString(16)));
}

if (go) {
  meteorits[i].material.color.setHex(0x6a6a6a);
}
```

- *Umso mehr Punkte umso schneller werden die Meteoriten*
- *Im Start-/GameOver-Screen blinken die Meteoriten bunt; im Spiel-Modus sind sie grau*

Systemarchitektur Funktionen Sternenhimmelbewegung

```
function starMov() {
  for (i = 0; i < stars.length; i++) {
    //Sterne bewegen
    stars[i].position.z += 1.0;

    //Sterne verschieben wenn hinter Kamera
    if (stars[i].position.z >= 100) {
      stars[i].position.x = 3000 * (2.0 * Math.random() - 1.0);
      stars[i].position.y = 3000 * (2.0 * Math.random() - 1.0);
      stars[i].position.z = -3000;//(Math.random() * (100 - (-3000)) + (-3000));
    }
  }
}
```

- *Funktion gleicht Meteoritenbewegung*

Systemarchitektur Funktionen Schussfunktion (Projektile)

```
var bullets = [];

function shoot(){
  //Projektile erstellen
  var bulletGeo = new THREE.SphereGeometry(1, 32, 32);
  var bulletMat = new THREE.MeshBasicMaterial({color: 0xff0000});
  var bullet = new THREE.Mesh(bulletGeo, bulletMat);

  //Aus dem Bug schießen
  bullet.position.x = ship.position.x;
  bullet.position.y = ship.position.y;
  bullet.position.z = ship.position.z - 11;
  if (bullets.length <= 2) {
    scene.add(bullet);
    bullets.push(bullet);
  }
}
```

- *Funktion erstellt Projektil-Objekte (nicht mehr als 3)*
- *Positioniert Projektil-Objekte vor den Bug des Raumschiffes*
- *Werden in einem Array gespeichert*

Systemarchitektur Funktionen Schussfunktion (Projektile)

```
function bulletMov(){
  //Projektile nach vorne bewegen
  for (i = 0; i < bullets.length; i++) {
    bullets[i].position.z -= 1.5;

    if (bullets[i].position.z < -150) {
      scene.remove(bullets[i]);
      bullets.shift();
      //console.log(bullets.toString());
    }

  }
}
```

- *Projektile des Arrays werden nach vorne verschoben (schießen)*
- *Ab einer bestimmten Entfernung verschwinden Projektile (shift)*

Systemarchitektur Funktionen Kollision

```
function collision(obj) {

  var originPoint = obj.position.clone();

  for (var vertexIndex = 0; vertexIndex < obj.geometry.vertices.length; vertexIndex++) {
    var localVertex = obj.geometry.vertices[vertexIndex].clone();
    var globalVertex = localVertex.applyMatrix4(obj.matrix);
    var directionVector = globalVertex.sub(obj.position);

    var ray = new THREE.Raycaster(originPoint, directionVector.clone().normalize());
    var collisionResults = ray.intersectObjects(meteorits);
    if (collisionResults.length > 0 && collisionResults[0].distance < directionVector.length()) {
      if (obj == ship && collisionResults[0].object.visible == true) {shipHit()}
      else {bulletHit(collisionResults[0].object)}
    }
  }
}
```

- *Kollisionserkennung mit Hilfe des Three.js Objekts Raycaster*
- *Quelle: https://github.com/stemkoski/stemkoski.github.com/blob/master/Three.js/Collision-Detection.html*
- *für eigene Bedürfnisse angepasst*

Systemarchitektur Funktionen Punktezähler

```
function scores() {
  score += 0.25;

  document.getElementById('score').innerHTML = Math.round(score/10) * 10;

  if (highscore < score) {
    highscore = score;
    document.getElementById('highscore').innerHTML = "highscore: " + (Math.round(highscore/10) * 10);
  }
}
```

- *ca. pro Sekunde 10 Punkte*
- *die Punkte steigen nur in 10er Schritten*
- *Highscore wird angepasst wenn der aktuelle Punktestand größer ist als der aktuelle Highscore*

Zeitplan vs. Realität

Plan KW	Aufgabe	Realität KW
26	Erstellung & Vorstellung Planungspräsentation (24.06.15)	26
31	Einarbeitung Three.js + Spielkonzept	31
32	Steuerung Held/Raumschiff + Hindernisse	33
33	Skybox, Kollisionsverhalten (z.B. ~~Lebensabzug~~, ~~*Explosion*~~) + **MS1**	34
34	*Raumschiff-Waffen (schießen), Hindernisse (abschießen),* Punktezähler ~~*(Website mit Highscore)*~~	38
35	Modellierung (Blender)	X

Zeitplan vs. Realität

Plan KW	Aufgabe	Realität KW
36	Einbau Modelle in Spielumgebung, eventuell Anpassung der Collider + **MS2**	X
37	Bugfixing/Feinheiten	40
38	Javascript-Optimierung wegen großer Rechenlast durch Three.js	X
39	Finale Präsentation & Highscore knacken ;-) + **MS3**	40
40	Endabgabe (02.10.15)	40

Aufgabenverteilung/Aufwand

Aufgabe	Plan	Realität
Erstellung & Vorstellung Planungspräsentation (24.06.15)	5h	5h
Einarbeitung Three.js + Spielkonzept	15h	30h
Steuerung Held/Raumschiff + Hindernisse	15h	10h
Skybox, Kollisionsverhalten (z.B. Game Over bei Zusammenstoß) + **MS1**	20h	35h
Raumschiff-Waffen (schießen), Hindernisse (abschießen), Punktezähler	15h	20h
Modellierung (Blender)	20h	X

Aufgabenverteilung/Aufwand

Aufgabe	Plan	Realität
Einbau Modelle in Spielumgebung, eventuell Anpassung der Collider + **MS2**	10h	X
Bugfixing/Feinheiten	xxh	5h
Javascript-Optimierung wegen großer Rechenlast durch Three.js	xxh	X
Finale Präsentation & Highscore knacken ;-) + **MS3**	5h	8h
Endabgabe (02.10.15)		

Vielen Dank für Ihre Aufmerksamkeit!

Stefanie Zahn
02.10.2015