Design and simulation of Geyser explosions through Computer Graphics

Francesco Lombardi, Alessandro Sestini

08/06/2017

Università degli Studi di Firenze
Dipartimento di Ingegneria dell'Informazione

Computer Graphics & 3D

Introduction

Introduction

Idea

Realization of Geyser's natural phenomenon caused by a water explosion through WebGL and Three.js technologies.

Introduction

Project Aim

- Simulation of the **physics** of water and steam particles
- Description of the motion of water particles following a **parabolic trajectory**
- Description of the **randomic** motion of steam particles in 3D space
- Creation of a pseudo-realistic scene

Scene

Scene

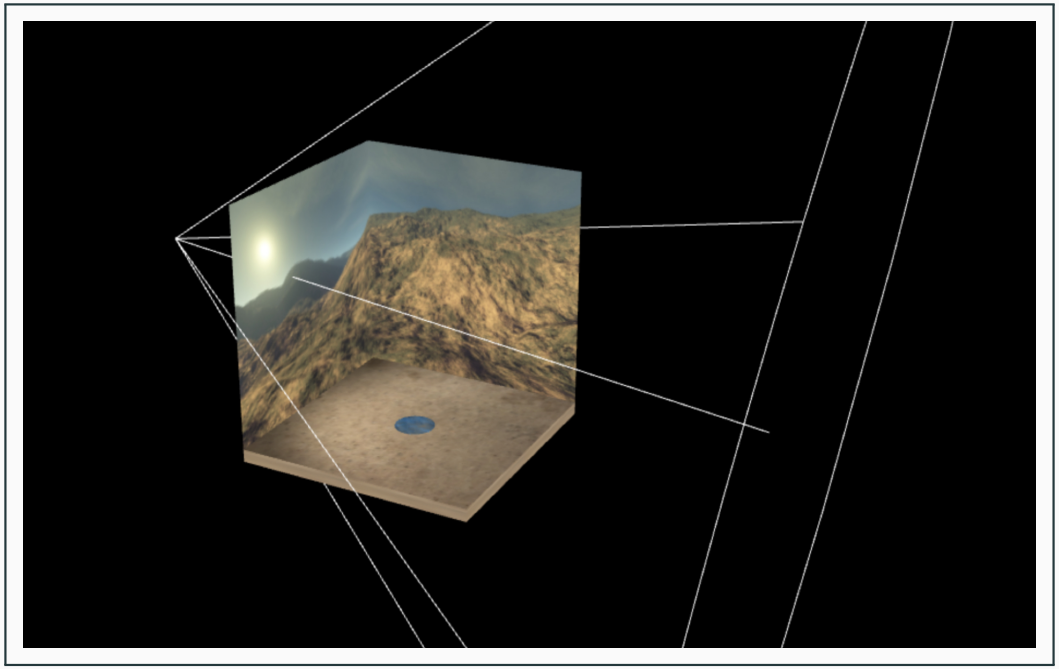
Environment Building

- Creation of a **cubeGeometry** with different textures for each face
- Rendering back-face culling: **back-side**
- Creation of the **cubic ground** applying a unique texture for all faces

Scene

Lights and Materials

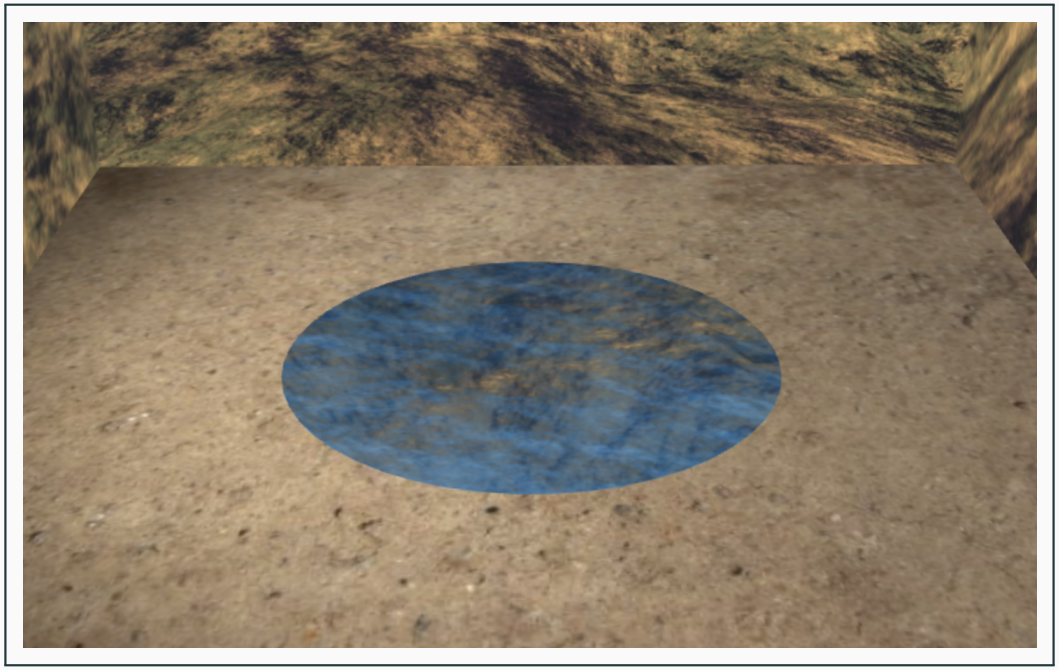
- Application of a white **Ambient Light**
- Addition of a **Spot Light** simulating sun light
- Application of a **Shiny** material to the ground in order to reflect incoming light



Scene

Water Mirror

- Realization of reflecting water through Three.js library with a custom shader

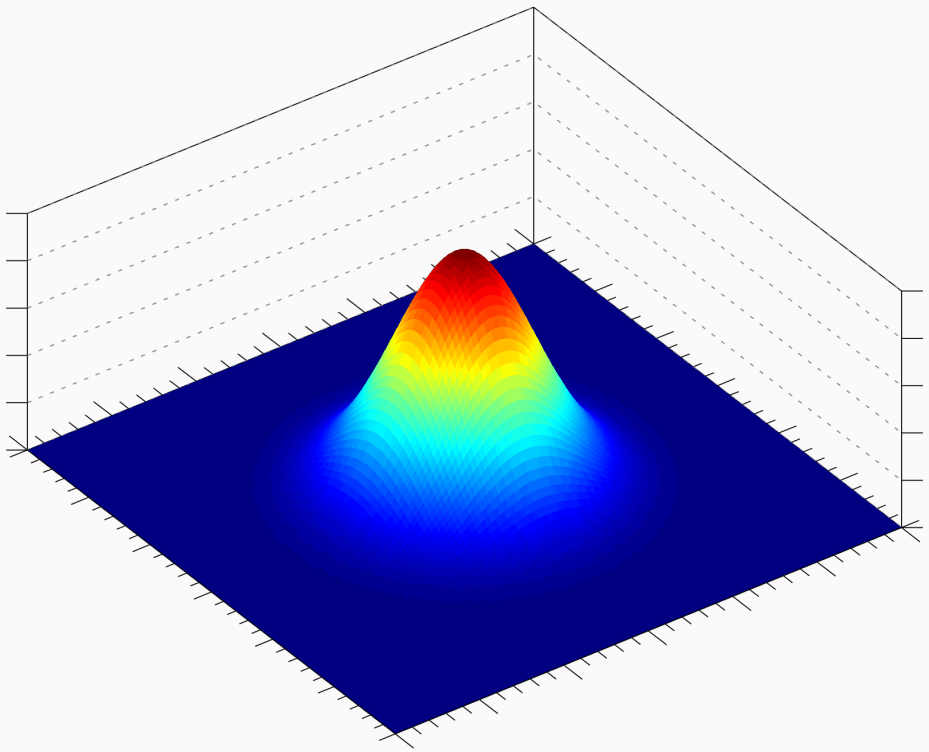


Physics

Particles Creation (for both water and steam)

Mapping particles position in 2D space by:

- Splitting a **Gaussian Function** along y-axis in sub-intervals creating concentric circles in the xz-plane

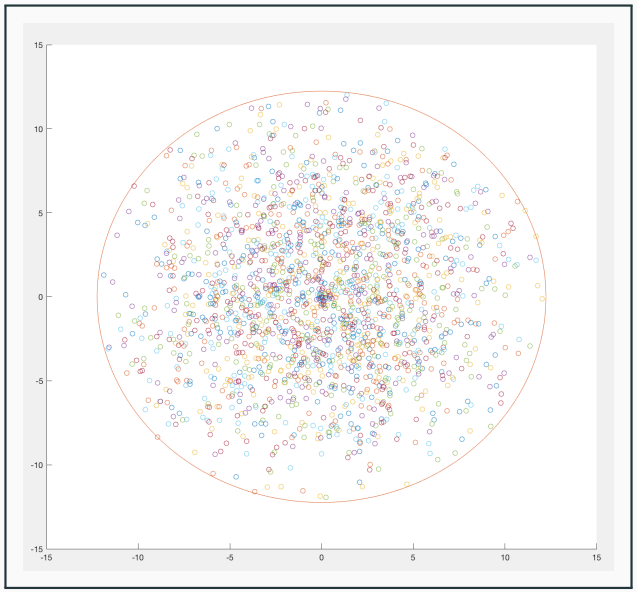


Physics - Particles Creation

- Arranging points in **random positions** all over the circles following the relation:

$$\begin{cases} x = R\,cos(\theta) \\ z = R\,sin(\theta) \end{cases}$$

- Pushing the xyz-coordinates in 2 different **Attribute Buffers** that are transferred to the shaders



Physics - Particles Creation

- Creating another Attribute Buffer with:
  - random value **v0y** for the initial speed along the y-axis
  - random value **v0r** for the initial speed along the xz-plane
  - the **radial direction** for each water particle in the xz-plane, given by the angle θ seen before
  - the direction for each steam particle in the xz-plane, given by a random value **between** $\omega \in [0, 2\pi]$

Physics - Particles Creation

- Creating **Uniform Variables** for each type:
  - **time** value t, inizialized to 0
  - **texture** images
  - default value of **opacity**
  - default value of **dimension**

Vertex Shader - Water

- Attribute Vector3 for initial **positions** (x, y and z) and **movements** ($v0y$, $v0r$ and θ)
- Uniform **float t** for time and uniform **float pointDim** for all particle dimensions
- The new position of a particle is computed following the law of **projectile motion**, with time value t upgraded every frame:
  - **costant** speed in the xz plane
  - uniformly **accelerated** motion along the y axis

$$\begin{cases} x(t) = v0r * cos(\theta) * t + x_0 \\ z(t) = v0r * sin(\theta) * t + z_0 \\ y(t) = -1/2 * g * t^2 + v0y * t + y_0 \end{cases}$$

Physics - Vertex Shader - Water

Code

```
attribute vec3 movements;

uniform float t;
uniform float pointDim;
uniform float limitXZ;

void main()
{

    float theta = movements.y;
    float v0r = movements.x;
    float v0y = movements.z;

    float v0x = v0r*cos(theta);
    float v0z = v0r*sin(theta);

    vec3 p = position;

    p.x += v0x*t;
    p.y += -0.5*9.8*80.0*t*t+v0y*t;
    p.z += v0z*t;

    if(p.y < -1.0 || abs(p.x) > limitXZ || abs(p.z) > limitXZ){
        p.x = -0.1;
        p.y = -0.1;
        p.z = -0.1;
    }

    vec4 mvPosition = modelViewMatrix * vec4( p, 1.0 );
    gl_PointSize = pointDim * ( 300.0 / -mvPosition.z );
    gl_Position = projectionMatrix * mvPosition;

}
```

Vertex Shader - Steam

- Like for water particles, same Vector3 for inital **positions** (x, y and z) and same Vector3 for **movements** ($v0y, v0r$ and ω)
- Same uniform variables for **time** and point **dimensions**
- The new position of a particle is computed following a **costant motion** in all 3 directions

$$\begin{cases} x(t) = v0r * cos(\omega) * t + x_0 \\ z(t) = v0r * sin(\omega) * t + z_0 \\ y(t) = v0y * t + y_0 \end{cases}$$

Physics - Vertex Shader - Steam

Code

```
attribute vec3 movements;

uniform float pointDim;
uniform float t;
uniform float limitXZ;

void main()
{

    float omega = movements.x;
    float v0r = movements.y;
    float v0y = movements.z;

    vec3 p = position;

    p.x += v0r*t*cos(omega);
    p.y += (v0y*t);
    p.z += v0r*t*sin(omega);

    if(abs(p.x) > limitXZ || abs(p.z) > limitXZ || p.y > 1200.0){
        p.x = -0.1;
        p.y = -0.1;
        p.z = -0.1;
    }

    vec4 mvPosition = modelViewMatrix * vec4( p, 1.0 );
    gl_PointSize = pointDim * ( 300.0 / -mvPosition.z );
    gl_Position = projectionMatrix * mvPosition;

}
```

Fragment Shader

- Same fragment shader for each type
- Only uniform variables for **textures** and **opacity** are defined
- If the value of opacity is set to 0, then **discard** the points

Physics - Fragment Shader

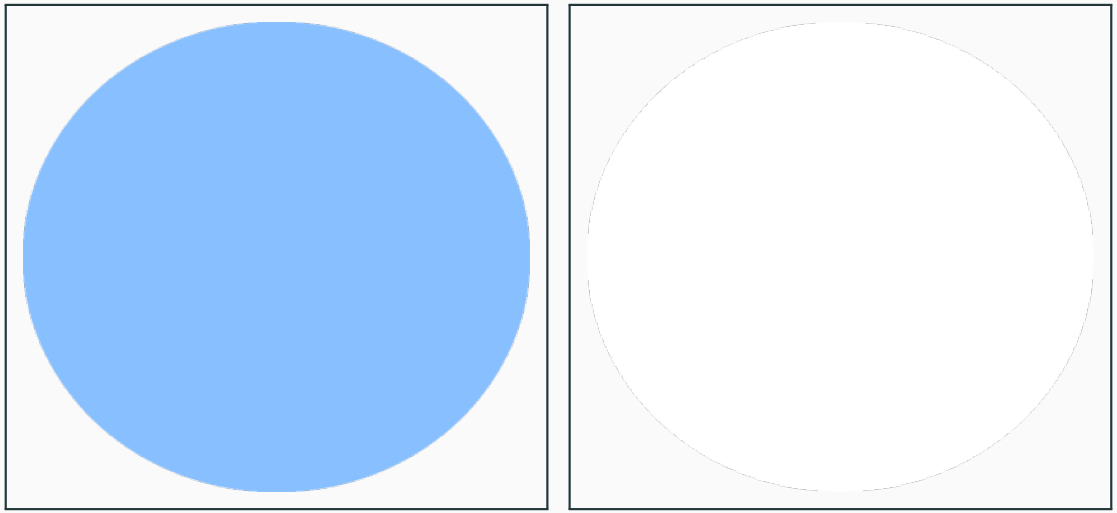
Code

```
uniform sampler2D texture_sampler;
uniform float opacity;


void main()
{

    if(opacity == 0.0){
        discard;
    } else {
        gl_FragColor = texture2D(texture_sampler, gl_PointCoord);
        gl_FragColor.a = opacity;
    }

}
```

Physics

Texture

- **Rounded textures** are used to simulate the effect of a more complex mesh
- **Additive Blending** technique is used to make the greater density areas gradually more white



Physics

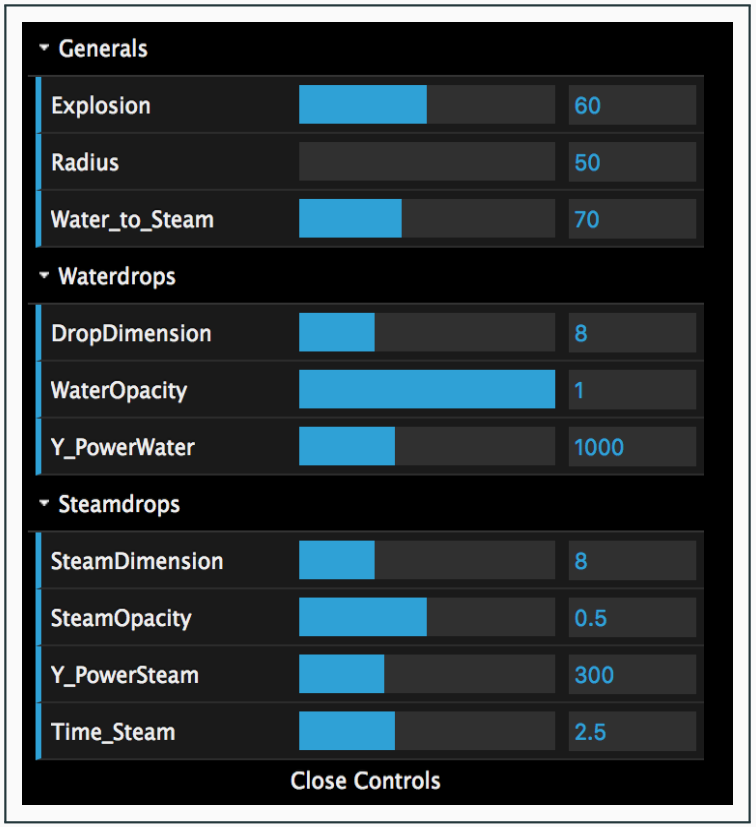
Render

When the space button is pressed, the render function distributes particles in the scene:

- particles movement is simulated by **upgrading the uniform variable t** in every frame and computing the new position for each point
- **new particles are created** frame by frame (so with t inizialized to 0) and they are 'exploded'
- the process is **repeated** until a threshold value is reached

Physics Controls

Some controls have been implemented in order to let the users be able to customize their experience:



Design and simulation of Geyser explosions through Computer Graphics

Francesco Lombardi, Alessandro Sestini

08/06/2017

Università degli Studi di Firenze
Dipartimento di Ingegneria dell'Informazione

Computer Graphics & 3D