MUSIC VIEWER - PARTICLES SYSTEM

COMPUTER GRAPHICS AND 3D

- Alessandro Minervini
- Alessandro Soci

IDEA

- Implement a **Music Viewer** made with particles systems

- Particles systems are divided into three groups:
  1. Bass particles system
  2. Treble particles system
  3. High particles system

- Use WebGL and shaders to achieve the behavior of particles systems following the background music

INTRODUCTION

To implement the music viewer

- The first step is analyze the background sound with a library called **Web audio Api**
- The second step is create the three particles systems
- The third step is connect the data of audio analysis with the behavior of particles systems through the shaders
- The fourth step is create the scene using another library, **Three JS**

SOUND ANALYSIS

- To analyze the sound the library used is **Web audio api**. This library contains methods to manage audio in a javascript context.
- The specific method is **.getByteFrequencyData(frequencyData);** situated in the loop function (render.js). It copies the current frequency data into a Uint8Array (FrequencyData, unsigned byte array) passed into it.
- This method is equivalent to a FFT (fast Fourier transform)
- FrequencyData length is half of FFT size

HOW IT WORKS

PROCEDURE:

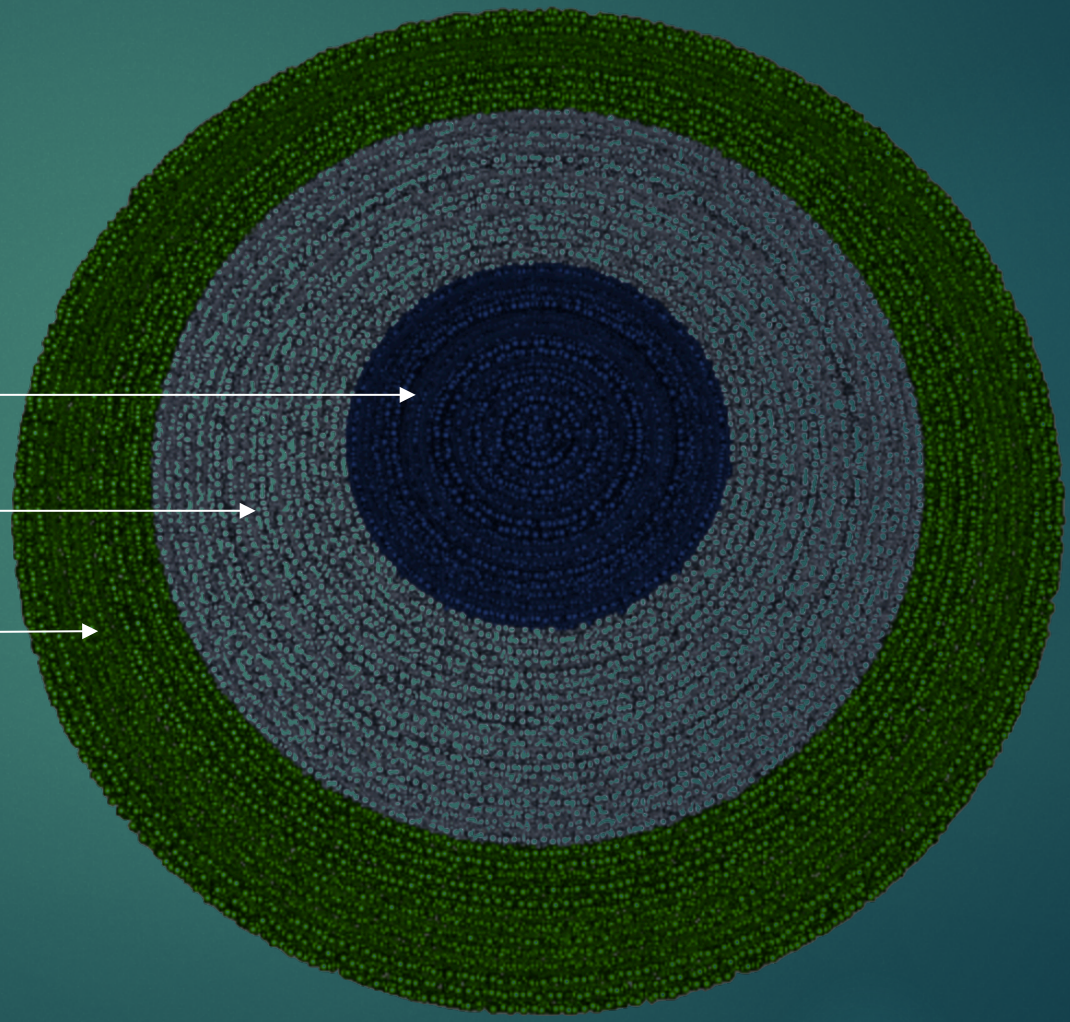
1. **AudioContext()** - Creates and returns a new AudioContext object
2. **var audio** – Upload the signal into this variable
3. **.createAnalyser()** – Method of AudioContext, creates a Analyzer object
4. **.connect** – Connect the analyzer to the signal target
5. **.fftSize** – Is an unsigned long value representing the size of the FFT to be used to determine the frequency domain
6. **.frequencyBinCount** – Set the length of FFT data values (half that of the FFT size)

```
var audioCtx = new AudioContext();
var audio = document.getElementById('signal');
var audioSrc = audioCtx.createMediaElementSource(audio);
var analyser = audioCtx.createAnalyser();

audioSrc.connect(analyser);

analyser.fftSize = 1024;
var bufferLength = analyser.frequencyBinCount;
var frequencyData = new Uint8Array(bufferLength);
```

INITIALIZE PARTICLES SYSTEMS

SCHEME:

- High particles
- Treble particles
- Bass particles



INITIALIZE PARTICLES SYSTEMS

- For each particles system the function **createVertexMovementsParticles()** sets the first position of particles, creating the central disk and the other two rings, calculating the vertices, and initializing the movements.
- **Vertices and movements are also added into the vertex shader** of respective particles system as attributes
- Next step is **define uniforms**. Here is passed the array frequencyData, containing the information of frequency **for updating the movements of particles**

VERTEX SHADER

Attributes:
- First position of vertices
- Movements (updating first positions of vertices)
- Size of particles
- Color

Uniforms:
- FrequencyData
- Opacity
- Texture

Other
- Index
- Smooth

UPDATING FREQUENCY DATA

- Into the loop function, **analyser.getByteFrequencyData(frequencyData)** calculates the values of frequency and put them into the array frequencyData every instant.
- The new frequencyData is passed to the uniform and updates the vertex shader
- FrequencyData.length is 512

UPDATING POSITIONS

- The shader calculates the position for each particle, and **uses this position as index to get the frequency value into frequencyData**
- This value will be the update position of respective particle as follows:
  - var index = ($\sqrt{x^2 + z^2}$);

    position.y = frequencyData[index];

PARTICLES FRAGMENT SHADER

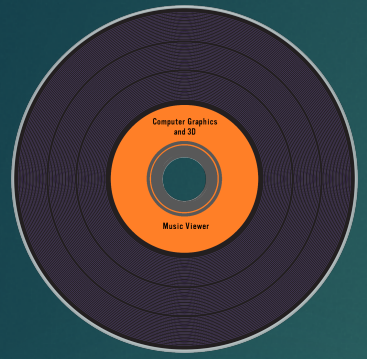
- Fragment shader is used to manage the color and texture shape of particles
- It's used a png image as texture
- The final results is the union of the color and texture
- The particles are spherical, centred in gl_PointCoord

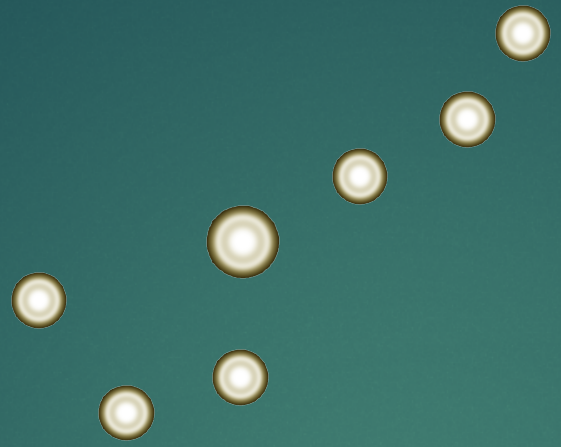
```
<script type="x-shader/x-fragment" id="fragment_particles">

    uniform sampler2D texture_sampler;
    uniform float opacity;
    varying vec4 vColor;


    void main(){

        vec2 circCoord = 2.0 * gl_PointCoord - 1.0;
        if (dot(circCoord, circCoord) > 1.0) {
            discard;
        }
        gl_FragColor = vColor;
        gl_FragColor = vColor*texture2D(texture_sampler, gl_PointCoord);
        gl_FragColor.a = opacity;
    }

</script>
```

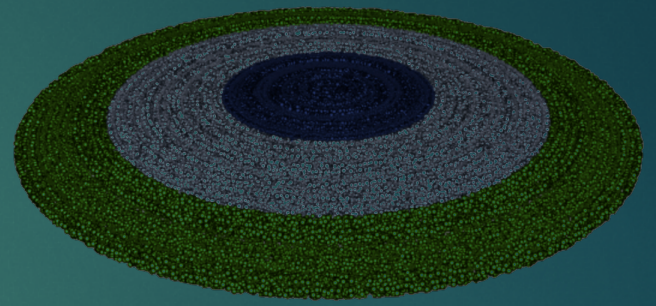
THREE.js

- Three.js is an Open Source javascript library that offers methods for interfacing WebGL core
- Three.js allows to create complex 3D animations and system
- It's necessary to create an object, defined by a Geometry and a Material. These have a lot of specialization.
- Let to control the 'camera' with the object OrbitControls

SCENE







VINYL

- CircleGeometry
- MeshPhongMaterial
  - Vinyl texture

STARS

- Geometry
  - Vector of position
- PointsMaterial
  - Color
  - Size
  - Map
  - Blending
  - trasparent

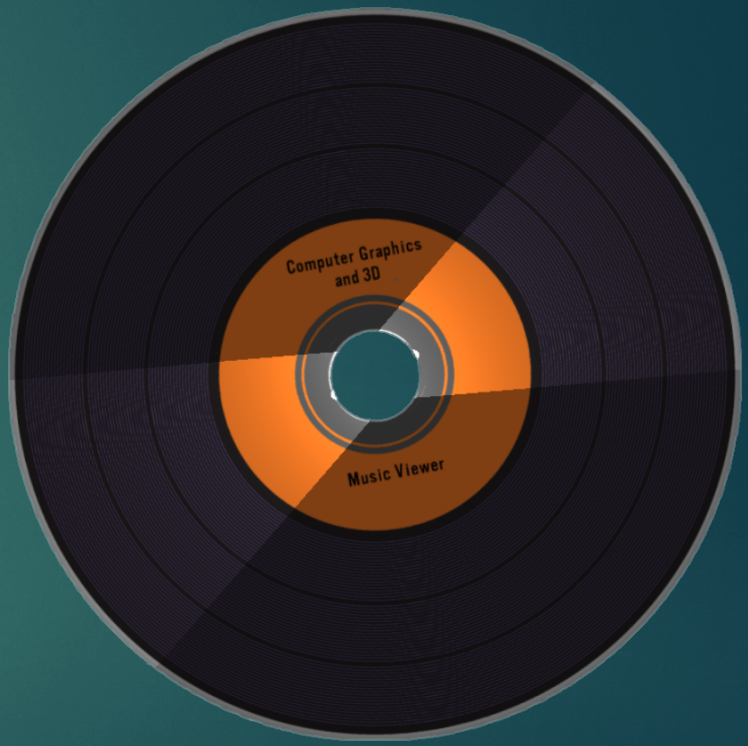
DISKS

- BufferGeometry
  - defined as a set of N vertices with *attributes*:
    - Position
    - customSize
    - boolColor
- ShaderMaterial
  - Texture
  - Vertex
  - Fragment

LIGHT

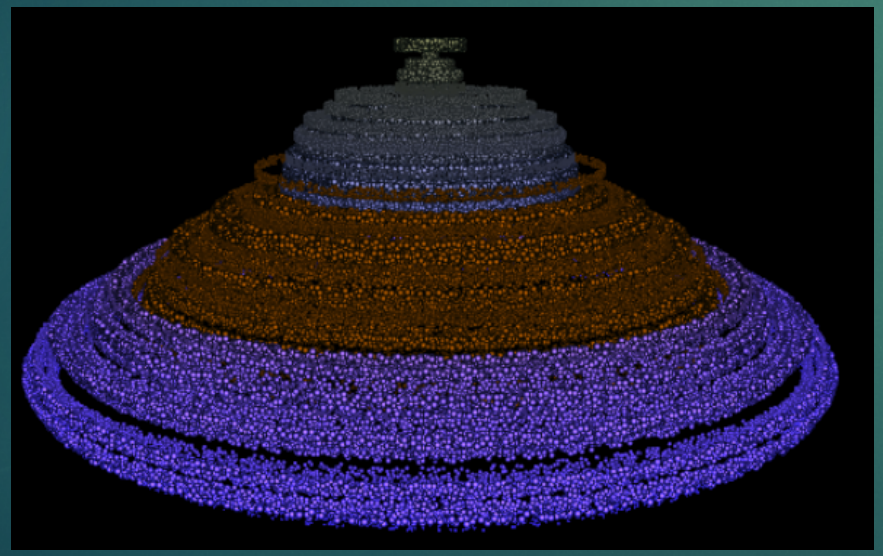
- There are 2 light to simulate the reflexes on vinyl

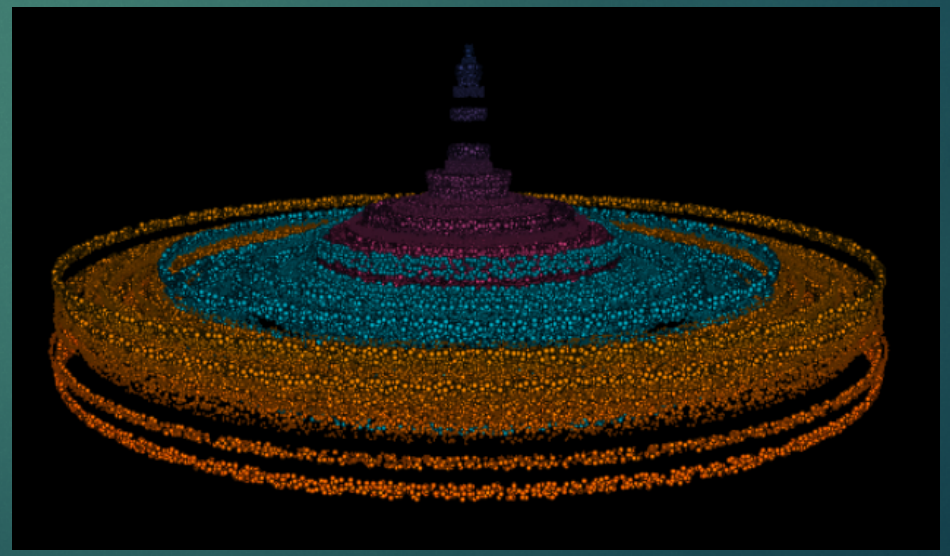
```
var spotLight1 = new THREE.SpotLight(0xffffff, 10, 200, 0.3, 0, 1.0);
spotLight1.castShadow = true;
spotLight1.position.set(40,10,-70);
spotLight1.angle = 0.4;
spotLight1.distance = 1100;
scene.add(spotLight1);

vinyl.receiveShadow = true;
```



COLORS

- There are 3 colors for each frequency
- The colors vary according to the frequency amplitude
- Possibilty to change the color of disks





```
if(boolColor == true){
    if(float(frequency[index]) == 0.0){
        vColor = vec4(customColor,opacity);
    }else{
        vColor.x = customColor.x + (-log2(float(frequency[index])/ma));
        vColor.y = customColor.y; //+ log2(float(x));
        vColor.z = customColor.z; //+ log2(float(z));
        vColor.a = opacity;
    }
}else{
    if(float(frequency[index]) == 0.0){
        vColor = vec4(customColor,opacity);
    }else{
        vColor.x = customColor.x;
        vColor.y = customColor.y;
        vColor.z = customColor.z + (-log2(float(frequency[index])/ma));
        vColor.a = opacity;
    }
}
```

CONTROLS

- With space (keycode 32) it's possible to stop/play the music

- The library used to modify some parameters is DatGui, to let the users be able to customize their experience:
    1. Particles size
    2. Particles number
    3. Change Color
    4. Change Musics
    5. Trasparency

CONCLUSION

The code is avaibile at this link: https://github.com/AlessandroMinervini/MusicViewerGL.

The result is possible to watch here.