Sección de Informática Gráfica VALENCIA | Computer Graphics Group
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Animación & IGU

Gráficos 3D en la web

Animación

Etapa de actualización

- Coherencia temporal
  - variables dependientes del tiempo transcurrido entre *frames*
  - cálculo del tiempo transcurrido
    - Date.now() : tiempo en msg desde el origen
    - THREE.Clock()
      - .getDelta() : tiempo entre llamadas en sg
    - render(time) : tiempo en msg desde el origen provisto por requestAnimationFrame()
- Función update()
  - v += inc_v(t);

inc_angulo = velocidad_angular * tiempo_trancurrido

```
var antes = Date.now();
function update()
{
    var ahora = Date.now();
    angulo += Math.PI/30 * (ahora - antes)/1000;
    antes = ahora;
    paraguas.rotation.y = angulo;
}

function render()
{
    requestAnimationFrame( render );
    update();
    renderer.render( scene, camera );
}
```

Interpolación

- animación = cambio = evolución
- evolución de un atributo
  - valor inicial y final
  - tiempo de evolución
  - forma matemática (función temporal del atributo)
  - muestreo automático: técnica de inbetweening
- Tween.js
  - https://github.com/tweenjs/tween.js
  - Motor de interpolación de uso sencillo
  - **Directamente integrable con three.js**

variable a interpolar

valor final de los atributos

```
var coords = { x: 0, y: 0 };
var tween = new TWEEN.Tween(coords)
    .to({ x: 100, y: 100 }, 1000)
    .onUpdate(function() {
        console.log(this.x, this.y);
    })
    .start();

requestAnimationFrame(animate);

function animate(time) {
    requestAnimationFrame(animate);
    TWEEN.update(time);
}
```

tiempo de interpolación

Interpolación

- TWEEN.Tween(objeto)
  - Construye un interpolador
  - Métodos principales:
    - .to(objeto, duración_msg)
    - .start()
    - .stop()
    - .update()
    - .chain(tween_siguiente)
    - .repeat(veces_extra)
    - .easing(TWEEN.Easing.funcion)
  - Call backs:
    - .onStart(callback)
    - .onComplete(callback)
    - .onUpdate(callback)

```
function init() {

  position = {x: 100, y: 100, rotation: 0};
  target = document.getElementById('target');
  tween = new TWEEN.Tween(position)
    .to({x: 700, y: 200, rotation: 359}, 2000)
    .delay(1000)
    .easing(TWEEN.Easing.Elastic.InOut)
    .onUpdate(update);

  tweenBack = new TWEEN.Tween(position)
    .to({x: 100, y: 100, rotation: 0}, 3000)
    .easing(TWEEN.Easing.Elastic.InOut)
    .onUpdate(update);

  tween.chain(tweenBack);
  tweenBack.chain(tween);

  tween.start();

}
```

ejemplo

Interpolación

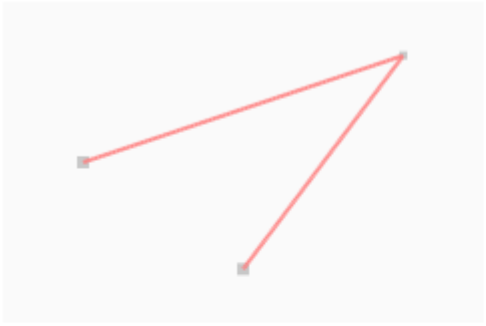
posición en la trayectoria origen-target respecto del tiempo
creciente: avanza; decreciente: retrocede

Interpolación

tipos de muestreo usando arrays

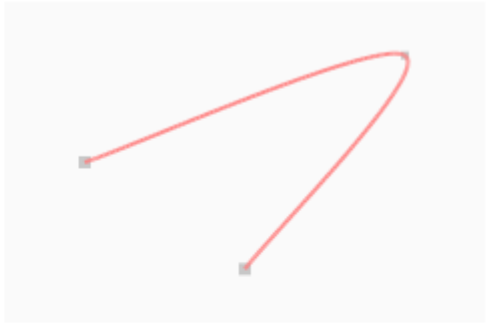
```
var tween = new TWEEN.Tween(relativeObj).to({ x: [0, -100, 100] });

tween.interpolation( TWEEN.Interpolation.Bezier );
```

Linear



Bezier

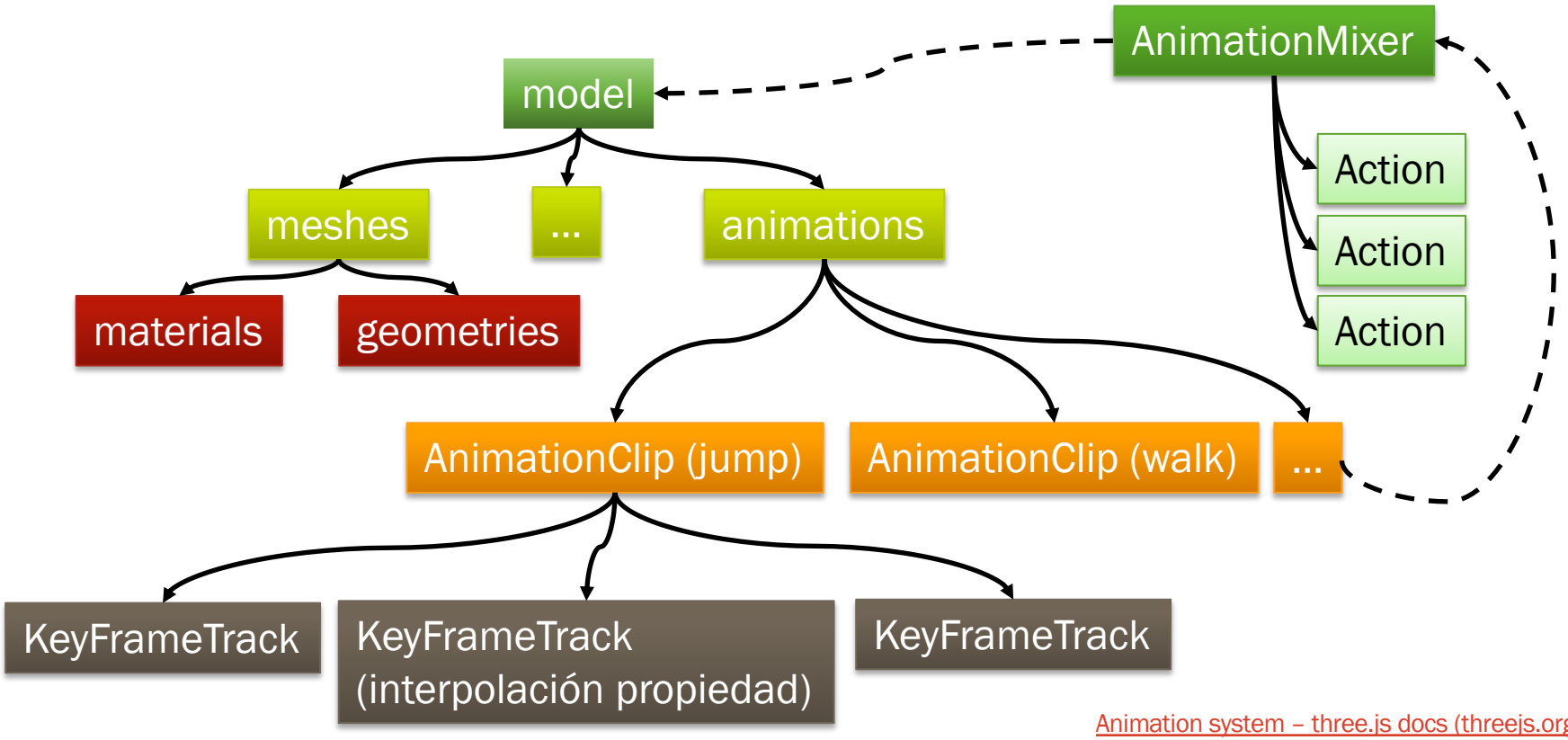
CatmullRom



start === end

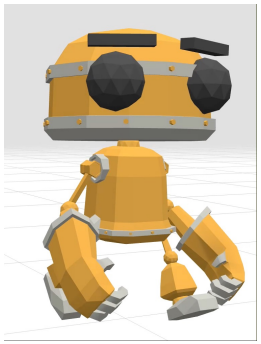
Animación en Threejs

- Basada en la especificación glTF
  - glTF Overview - The Khronos Group Inc
  - KhronosGroup/glTF-Tutorials: glTF Tutorials (github.com)



Animation system – three.js docs (threejs.org)

Animación en Threejs



```
let animatedObject ; // Object3D and descendants
let animations ;     // Array of AnimationClips
/*...*/
// Create an AnimationMixer and the actions list
const mixer = new THREE.AnimationMixer( animatedObject );
const actions = {};
animations.forEach(
    function(clip) {
    actions[clip.name] = mixer.AnimationClip(clip);
    }
)

// Play a specific animation
actions['dance'].play();

// Play all animations
for( let action in actions ) { actions[action].play(); };

// Update the mixer on each frame
function update () {
    mixer.update( deltaSeconds );
}
```

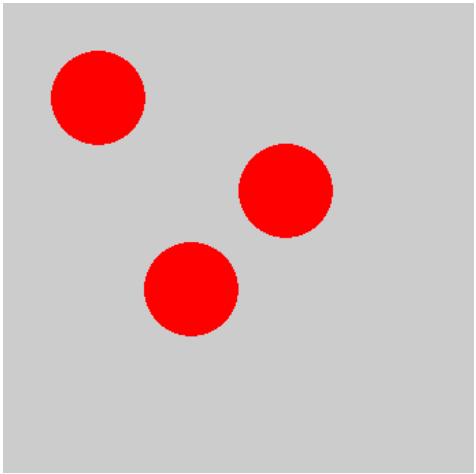
```
▼ Object i
  ▼ animations: Array(14)
    ▶ 0: AnimationClip {name: 'Dance', tracks: Array(14), duration: 3.333333253860473
    ▶ 1: AnimationClip {name: 'Death', tracks: Array(20), duration: 0.958333313465118
    ▶ 2: AnimationClip {name: 'Idle', tracks: Array(9), duration: 3.3333332538604736,
    ▼ 3: AnimationClip
        blendMode: 2500
        duration: 0.7083333134651184
        name: "Jump"
      ▼ tracks: Array(20)
        ▶ 0: NumberKeyframeTrack {name: 'Head_2.morphTargetInfluences', times: Float32Array(18), values: Float32Array(54), createInterpolant: ƒ}
        ▶ 1: NumberKeyframeTrack {name: 'Head_3.morphTargetInfluences', times: Float32Array(18), values: Float32Array(54), createInterpolant: ƒ}
        ▶ 2: NumberKeyframeTrack {name: 'Head_4.morphTargetInfluences', times: Float32Array(18), values: Float32Array(54), createInterpolant: ƒ}
        ▶ 3: VectorKeyframeTrack {name: 'FootL.position', times: Float32Array(18), values: Float32Array(54), createInterpolant: ƒ}
        ▶ 4: QuaternionKeyframeTrack {name: 'FootL.quaternion', times: Float32Array(18), values: Float32Array(72), createInterpolant: ƒ}
```

nodo afectado

propiedad interpolada

Motores de física

- Durante la animación
  - los objetos interactúan
  - están sometidos a leyes físicas
- Motores de físicas
  - conjunto de primitivas
    - esferas, cajas, planos, etc
  - dinámica
    - sólido rígido usualmente
    - colisión, gravedad, fricción, etc
    - restricciones
- Ejemplos
  - ammo.js
  - cannon-es
  - Phisijs
  - Oimo.js
  - enable3d



Motores de física

- Cannon.js
  - Biblioteca de simulación de dinámica del sólido rígido independiente del motor de render
  - Mundo
    - Entorno de simulación y nodo raíz
    - Propiedades:
      - gravedad
      - paso de simulación (ms)
  - Cuerpo rígido
    - Propiedades geométricas (forma)
    - Propiedades físicas (masa p.e.)
  - Restricciones
    - Limitan los grados de libertad
    - Conexiones entre cuerpos
    - Imposibilidad de penetración
  - cannon-es

```
import * as CANNON from 'cannon-es'

// Setup our physics world
const world = new CANNON.World({
  gravity: new CANNON.Vec3(0, -9.82, 0), // m/s²
})

// Create a sphere body
const radius = 1 // m
const sphereBody = new CANNON.Body({
  mass: 5, // kg
  shape: new CANNON.Sphere(radius),
})
sphereBody.position.set(0, 10, 0) // m
world.addBody(sphereBody)

// Create a static plane for the ground
const groundBody = new CANNON.Body({
  type: CANNON.Body.STATIC, // can also be achieved by setting the mass
to 0
  shape: new CANNON.Plane(),
})
groundBody.quaternion.setFromEuler(-Math.PI / 2, 0, 0) // make it face
up
world.addBody(groundBody)

// Start the simulation loop
function animate() {
  requestAnimationFrame(animate)

  world.fixedStep()

  // the sphere y position shows the sphere falling
  console.log(`Sphere y position: ${sphereBody.position.y}`)
}
animate()
```

(c) cannon-es

Motores de física

- Cannon.js
  - Conexión con Threejs
    - Cuerpo físico vs cuerpo visual
    - Copia propiedades físico -> visual

cannon

```
// Box
const shape = new CANNON.Box(new CANNON.Vec3(1, 1, 1))
body = new CANNON.Body({
  mass: 1,
})
body.addShape(shape)
body.angularVelocity.set(0, 10, 0)
body.angularDamping = 0.5
world.addBody(body)
```

threejs

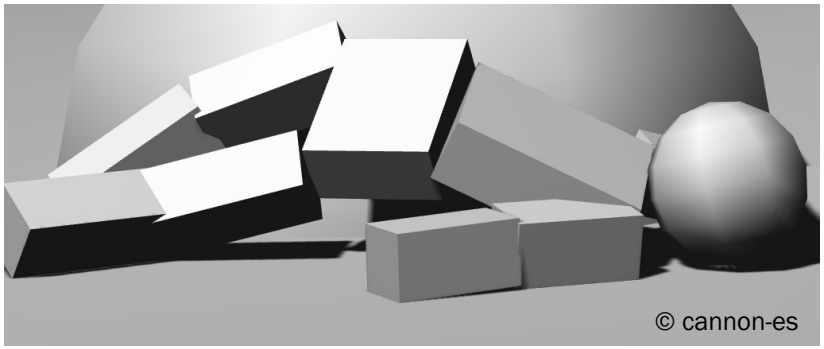
```
// Box
const geometry = new THREE.BoxBufferGeometry(2, 2, 2)
const material = new THREE.MeshBasicMaterial({ color: 0xff0000, wireframe: true })

mesh = new THREE.Mesh(geometry, material)
scene.add(mesh)
```

```
// Copy coordinates from cannon.js to three.js
mesh.position.copy(body.position)
mesh.quaternion.copy(body.quaternion)
```

cada frame

Motores de física

- Cannon.js
  - Uniones (constraints)
    - Cuerpos en contacto
    - Definición de la restricción

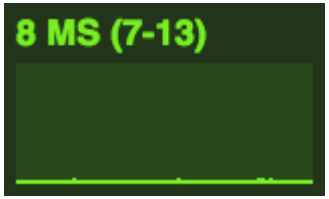


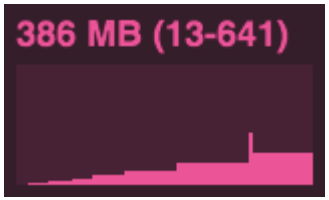

```
// Knee joints
const leftKneeJoint = new CANNON.ConeTwistConstraint(lowerLeftLeg, upperLeftLeg, {
  pivotA: new CANNON.Vec3(0, 0, lowerLegLength / 2),
  pivotB: new CANNON.Vec3(0, 0, -upperLegLength / 2),
  axisA: CANNON.Vec3.UNIT_Z,
  axisB: CANNON.Vec3.UNIT_Z,
  angle,
  twistAngle,
})
```

Estadísticas animadas

- stats.js
  - https://github.com/mrdoob/stats.js/
  - Biblioteca para muestra animada de estadísticas de rendimiento
  - FPS: frames por segundo
  - MS: milisegundos por frame
  - MB: Mbytes de memoria usada

```
var stats = new Stats();
stats.showPanel( 1 ); // 0: fps, 1: ms, 2: mb, 3+: custom
document.body.appendChild( stats.dom );

function animate() {

    stats.begin();

    // monitored code goes here

    stats.end();

    requestAnimationFrame( animate );

}
```

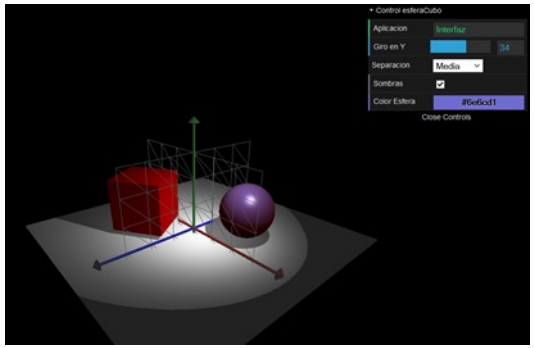




Interfaz

Interfaz gráfica de usuario

- Biblioteca ligera de controles gráficos para que el usuario pueda modificar parámetros de la escena
- dat.gui
- lil-gui




```
import GUI from 'lil-gui';

const gui = new GUI();

const myObject = {
    myBoolean: true,
    myFunction: function() { ... },
    myString: 'lil-gui',
    myNumber: 1
};

gui.add( myObject, 'myBoolean' );  // Checkbox
gui.add( myObject, 'myFunction' ); // Button
gui.add( myObject, 'myString' );   // Text Field
gui.add( myObject, 'myNumber' );   // Number Field
```