Passeio Virtual 3D em WebGL Puro

Categoria: Passeio 3D

Eduardo Matias Lucas Emanuel Murilo Sousa

2026

O que entregamos

- Um passeio virtual 3D em WebGL puro (sem bibliotecas graficas de alto nivel)
- Cena: hall principal com itens de exibicao + modelos OBJ carregados
- Interacao: camera em primeira pessoa, colisao e fisica basica (pulo)
- Render: pipeline 3D com projeção perspectiva, clipping, z-buffer e Phong

Checklist dos requisitos

- Camera com projeção perspectiva: `mat4.perspective(..., near, far)` em `main.js`
- Phong no fragment shader: `fragment.glsl` (ambient + diffuse + specular)
- Luz com movimentacao: `uLightPos` variando com `sin/cos` no loop
- Objeto animado por transformacao: rotacoes por `rotSpeed` nos itens de exibicao
- Textura e UV: piso e parede com `aTexCoord` + `texture2D`
- Cor solida: `uObjectColor` + flag `uUseTexture`
- Cena feita em WebGL puro: buffers, atributos e uniforms na mao
- OBJ loader manual: `obj_loader.js` (parse linha a linha)

Pipeline 3D (visao de alto nivel)

- Entrada: vertices em espaco do objeto (atributos)
- Vertex shader: aplica Model, View, Projection
- Clipping: recorte no clip space (fora do frustum sai)
- Rasterizacao: triangulos viram fragmentos (pixels candidatos)
- Fragment shader: calcula cor (Phong + textura)
- Z-buffer: resolve superficies ocultas (depth test)

$$p_{clip} = M_{proj} \cdot M_{view} \cdot M_{model} \cdot p_{obj}$$

Projecao, View, Z-buffer no loop (main.js)

```
gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);
gl.clearColor(0.05, 0.05, 0.1, 1.0);
gl.enable(gl.DEPTH_TEST);
gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

const projectionMatrix = mat4.create();
mat4.perspective(
  projectionMatrix,
  45 * Math.PI / 180,
  gl.canvas.width / gl.canvas.height,
  0.1,   // near (clipping)
  200.0  // far  (clipping)
);

const viewMatrix = mat4.create();
vec3.add(target, cameraPos, cameraFront);
mat4.lookAt(viewMatrix, cameraPos, target, cameraUp);
```

Clipping 3D (por que near/far importam)

- O frustum da camera e definido por FOV, aspect, near e far
- Tudo fora do frustum e recortado no clip space
- near muito pequeno piora precisao do depth (z-fighting)
- no projeto: near=0.1 e far=200.0 (equilibrio para a escala do hall)

Camera FPS (yaw/pitch + WASD)

- Controlamos orientacao por yaw/pitch (mouse)
- Geramos o vetor `front` e recomputamos `right` com produto vetorial
- Movimento: W/S no `front`, A/D no `right`

Yaw/Pitch e vetores da camera (camera.js)

```
const radYaw = this.yaw * Math.PI / 180;
const radPitch = this.pitch * Math.PI / 180;

front[0] = Math.cos(radYaw) * Math.cos(radPitch);
front[1] = Math.sin(radPitch);
front[2] = Math.sin(radYaw) * Math.cos(radPitch);

vec3.normalize(this.front, front);

// Right = Front x Up (produto vetorial)
vec3.cross(this.right, this.front, this.up);
vec3.normalize(this.right, this.right);
```

Movimento WASD (camera.js)

```
if (this.keys['w']) {
  vec3.scale(temp, this.front, this.speed);
  vec3.add(this.position, this.position, temp);
}
if (this.keys['a']) {
  vec3.scale(temp, this.right, this.speed);
  vec3.sub(this.position, this.position, temp);
}
this.position[1] = 1.6; // trava no chao
```

LookAt: estado atual e opcao manual

- Estado atual do projeto: `mat4.lookAt` (glMatrix) gera a View Matrix
- Se o professor exigir LookAt manual, basta trocar por uma funcao propria
- No proximo slide: LookAt manual pronto para colar

LookAt manual

```
// eye, center, up: vec3 (arrays length 3)
// out: mat4 (Float32Array length 16)
function lookAtManual(out, eye, center, up) {
  // z = normalize(eye - center)
  let zx = eye[0] - center[0], zy = eye[1] - center[1], zz = eye[2] - center[2];
  let zlen = Math.hypot(zx, zy, zz) || 1;
  zx/=zlen; zy/=zlen; zz/=zlen;

  // x = normalize(up x z)
  let xx = up[1]*zz - up[2]*zy;
  let xy = up[2]*zx - up[0]*zz;
  let xz = up[0]*zy - up[1]*zx;
  let xlen = Math.hypot(xx, xy, xz) || 1;
  xx/=xlen; xy/=xlen; xz/=xlen;

  // y = z x x
  let yx = zy*xz - zz*xy;
  let yy = zz*xx - zx*xz;
  let yz = zx*xy - zy*xx;

  // Matriz view (col-major)
  out[0]=xx; out[4]=xy; out[8] =xz; out[12]=-(xx*eye[0]+xy*eye[1]+xz*eye[2]);
```

Transformacoes 3D e animacao

- Transformacoes com matrizes 4x4: translacao, rotacao e escala
- Animacao no tempo: alterar um angulo e reconstruir o Model Matrix
- No projeto: itens de exibicao rotacionam com `rotSpeed`

$$M_{model} = T \cdot R \cdot S$$

Dados dos itens da exibicao (main.js)

```
const exhibitionItems = [
  { model: 'statue', pos: [-14, 0.45, -12],
    scale: [0.24, 0.24, 0.24], rotSpeed: 0.25,
    color: [0.9, 0.9, 0.92] },
  { model: 'moon', pos: [-7, 0.55, -12],
    scale: [0.55, 0.55, 0.55], rotSpeed: -0.2,
    color: [0.82, 0.86, 0.96] }
];
```

Colisao e fisica basica (main.js)

```
if (!isGrounded) velocity[1] += gravity * deltaTime;

const newPos = [
  cameraPos[0] + velocity[0] * deltaTime,
  cameraPos[1] + velocity[1] * deltaTime,
  cameraPos[2] + velocity[2] * deltaTime
];

if (newPos[1] - playerHeight <= groundLevel) {
  newPos[1] = groundLevel + playerHeight;
  velocity[1] = 0;
  isGrounded = true;
}
```

Texturas e UV mapping

- UVs mapeiam cada vertice para um ponto da imagem 2D (0..1)
- Vertex shader repassa `aTexCoord` para o fragment shader
- Fragment shader usa `texture2D(uSampler, vTexCoord)`

UVs do cubo (geometry.js)

```
const texCoords = [
  // Frente (2 triangulos)
  0, 0,  1, 0,  1, 1,
  0, 0,  1, 1,  0, 1,
  // Tras (2 triangulos)
  0, 0,  0, 1,  1, 1,
  0, 0,  1, 1,  1, 0
];
```

Bind de buffers para atributos (utils.js)

```
const positionLoc = gl.getAttribLocation(program, 'aPosition');
gl.bindBuffer(gl.ARRAY_BUFFER, buffers.position);
gl.vertexAttribPointer(positionLoc, 3, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(positionLoc);

const texCoordLoc = gl.getAttribLocation(program, 'aTexCoord');
gl.bindBuffer(gl.ARRAY_BUFFER, buffers.texCoord);
gl.vertexAttribPointer(texCoordLoc, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(texCoordLoc);
```

Sombreamento local (Phong) vs global

- Aulas discutem coloracao local e global
- No projeto usamos iluminacao local (Phong), mais barata e adequada ao realtime
- Componentes: ambiente, difusa (Lambert) e especular (Phong)

Vertex shader: pipeline e normal matrix (vertex.glsl)

```
vPosition = vec3(uModelMatrix * vec4(aPosition, 1.0));
vNormal   = normalize(uNormalMatrix * aNormal);
vTexCoord = aTexCoord;

gl_Position =
  uProjectionMatrix * uViewMatrix * uModelMatrix * vec4(aPosition, 1.0);
```

Fragment shader: Phong completo + textura/cor (fragment.glsl)

```
vec3 ambient = 0.15 * uLightColor;

vec3 norm = normalize(vNormal);
vec3 lightDir = normalize(uLightPos - vPosition);
float diff = max(dot(norm, lightDir), 0.0);
vec3 diffuse = diff * uLightColor;

vec3 viewDir = normalize(uViewPos - vPosition);
vec3 reflectDir = reflect(-lightDir, norm);
float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32.0);
vec3 specular = 0.5 * spec * uLightColor;

vec4 baseColor = uUseTexture ? texture2D(uSampler, vTexCoord)
                             : vec4(uObjectColor, 1.0);

gl_FragColor = vec4(ambient + diffuse + specular, 1.0) * baseColor;
```

Luz movel (main.js)

```
lightPositions[0] = [
  Math.sin(now) * 8,
  5.0,
  Math.cos(now) * 8
];

setUniform3f(gl, program, "uLightPos", lightPositions[0]);
setUniform3f(gl, program, "uViewPos", cameraPos);
```

OBJ loader manual (parse linha a linha)

- Leitura de `v`, `vn`, `vt` e `f`
- Faces sao trianguladas em fan
- Cache de vertices evita duplicacao e gera indices
- Se nao vier normal no OBJ, calculamos por produto vetorial

Parser OBJ: switch por token (obj_loader.js)

```
switch (parts[0]) {
  case 'v':
    positions.push([+parts[1], +parts[2], +parts[3]]);
    break;
  case 'vn':
    normals.push([+parts[1], +parts[2], +parts[3]]);
    break;
  case 'vt':
    uvs.push([+parts[1], +parts[2]]);
    break;
  case 'f':
    processFace(parts.slice(1), data);
    break;
}
```

Triangulacao em fan (obj_loader.js)

```
for (let i = 1; i < faceIndices.length - 1; i++) {
  data.finalIndices.push(
    faceIndices[0],
    faceIndices[i],
    faceIndices[i + 1]
  );
}
```

Carregando modelos OBJ do acervo (main.js)

```
const statueData = await loadOBJ('assets/statue.obj');
if (statueData) objModels.statue = initOBJBuffers(gl, statueData);

const moonData = await loadOBJ('assets/moon.obj');
if (moonData) objModels.moon = initOBJBuffers(gl, moonData);
```

Arquitetura do projeto (modularizacao)

- `main.js`: loop, camera/physics, cena, uniforms, draw
- `shader_loader.js`: carrega, compila e linka shaders
- `geometry.js`: geometria gerada em codigo (cubo) + UVs
- `utils.js`: buffers, uniforms, texturas, bind de atributos
- `obj_loader.js`: parser OBJ manual
- `vertex.glsl / fragment.glsl`: pipeline + Phong

Demo

Demonstracao do passeio 3D