”Tetris 3D”: Trabalho final de Computação Gráfica

Daniele Karoline Carvalho Rosa, Mauricío Catanio

[1]Universidade Federal da Fronteira Sul (UFFS)
SC-484, Km 02 - Fronteira Sul, Chapecó - SC, 89815-899

[2]Ciência da Computação
Universidade Federal da Fronteira Sul (UFFS)

carvalho.danielekr@gmail.com, mauricatanio@yahoo.com.br

Resumo. *”Ela é a pura representação visual da matemática.”Assim define o professor George Francis, do Departamento de Matemática da Universidade de Illinois, a computação gráfica. Com o avanço da computação, internet e linguagens de programação, projetar com Computação Gráfica (CG), se tornou mais prático. Este trabalho tem como objetivo, apresentar o desenvolvimento de um jogo similar ao Tetris com CG, utilizando ThreeJS e outras tecnologias para modelagem 3D, para a matéria de Computação Gráfica, da graduação de Ciência da Computação, UFFS.*

1. Introdução

A Computação Gráfica (CG) é uma área da Ciência da Computação que se dedica ao estudo e desenvolvimento de técnicas e algoritmos para a geração (síntese) de imagens através do computador. Atualmente, a CG está presente em quase todas as áreas do conhecimento humano, desde o projeto de um novo modelo de automóvel até o desenvolvimento de ferramentas de entretenimento, entre as quais os jogos eletrônicos.[Manssour] A amigabilidade proporcionada pelas interfaces gráficas, cada vez mais sofisticadas e ao mesmo tempo mais naturais, faz com que a Computação Gráfica esteja cada vez mais próxima dos “usuários” que, muitas vezes, nem se darão conta de que estão diante de um um produto desta ciência. [Santos] Para desenvolver o trabalho proposto, utilizamos o Three.js. Three.js é uma biblioteca 3D que tenta facilitar ao máximo a obtenção de conteúdo 3D em uma página da web. Three.js é frequentemente confundido com WebGL, pois na maioria das vezes, mas nem sempre, three.js usa WebGL para desenhar 3D. WebGL é um sistema de nível muito baixo que desenha apenas pontos, linhas e triângulos . Fazer algo útil com WebGL geralmente requer um pouco de código e é aí que entra o three.js. Ele lida com coisas como cenas, luzes, sombras, materiais, texturas, matemática 3D, todas as coisas que você mesmo teria que escrever se você usasse o WebGL diretamente.[Three.js] O Tetris é um jogo clássico, criado na década de 80, onde se tinha pouca técnologia gráfica. A proposta desse trabalho é criar um jogo parecido com Tetris, utilizando modelagem 3D.

2. Objetivos Específicos

Criar um jogo similar a Tetris por meio da programação com Javascript, utilizando a biblioteca Three.js e outras ferramentas. Espera-se a finalização do trabalho proposto, de forma satisfatória, com suas funcionalidades pricipais e lógica estruturada.

3. Técnicas Propostas

Para o desenvolvimento do trabalho, será utilizada as ferramentas de programação web, tais quais HTML, CSS e Javasript, além da biblioteca base para o avanço e conclusão do projeto, a Three.js e também, as bibliotecas de referência para modelagem 3D. Da biblioteca Three.js para criar uma cena 3D interativa no navegador. Inicia-se a biblioteca Three.js e o componente OrbitControls para permitir a manipulação da câmera. Configurou-se a cena, a câmera e o renderizador, posicionando a câmera e anexando o elemento de renderização ao corpo do documento.

4. Funções Principais do Código

Figura 1. Trecho do código da criação das cenas

```
function init() {
    createThreeSidedGrid();
    addLights();

}
document.getElementById('startGameButton').addEventListener('click', function() {

    setInterval(updatePiecePosition, 500);
    animate();
    this.disabled = true;
});

init();

function animate() {
    requestAnimationFrame(animate);
    if (!gameOver) {
        dropPiece();
    }
    renderer.render(scene, camera);
}


function addLights() {
    const ambientLight = new THREE.AmbientLight(0xffffff, 0.5);
    scene.add(ambientLight);

    const directionalLight = new THREE.DirectionalLight(0xffffff, 0.5);
    directionalLight.position.set(10, 15, 10);
    scene.add(directionalLight);
}
```

Configura o ambiente básico para uma aplicação 3D usando Three.js, incluindo a cena, câmera, renderizador, e controles de órbita, além de preparar variáveis para controle de peças, detecção de colisões e estado do jogo. A função *init* configura o ambiente do jogo, adicionando iluminação e iniciando um intervalo para atualizar a posição das peças, além de iniciar o loop de animação. A função *animate* mantém o loop de animação, atualizando a posição das peças se o jogo não tiver terminado e renderizando a cena a cada frame.*Three.Js*. Necessáiria para a utilização da biblioteca, animações e outros recursos.

Figura 2. Trecho do código da criação da grade e backgorund

```
function createThreeSidedGrid() {
    const size = 10;
    const divisions = 10;
    const color = new THREE.Color("gray");

    const group = new THREE.Group();
    group.name = "Grids"

    const baseGrid = new THREE.GridHelper(size, divisions, color, color);
    baseGrid.position.y = -size / 2;
    group.add(baseGrid);

    const leftGrid = new THREE.GridHelper(size, divisions, color, color);
    leftGrid.rotation.x = Math.PI / 2;
    leftGrid.rotation.z = Math.PI / 2;
    leftGrid.position.x = -size / 2;
    group.add(leftGrid);

    const backGrid = new THREE.GridHelper(size, divisions, color, color);
    backGrid.rotation.x = Math.PI / 2;
    backGrid.position.z = -size / 2;
    group.add(backGrid);

    scene.add(group);
}
```

Cria uma grade tridimensional composta por três grades perpendiculares entre si, utilizando a biblioteca Three.js. É renderizada na cena 3D, oferecendo uma referência visual útil para posicionamento e orientação de objetos no espaço tridimensional.

Figura 3. Trecho do código do geração das peças dentro do cenário e a movimentação no eixo Y

```
function dropPiece() {
    if (currentPiece === null) {
        currentPiece = getRandomPiece();
        currentPiece.position.y = 4.5;
        currentPiece.name = `Piece (${scene.children.length - 5})`;
        scene.add(currentPiece);
    }
}

function updatePiecePosition() {
    if (currentPiece && colisionEnabled) {
        currentPiece.position.y -= 1;

        if (colidesWithSceneObjects(currentPiece)) {
            currentPiece.position.y += 1;

            if (currentPiece.position.y > 4.4){
                console.log("Game Over");
                alert("Game Over! x.x")
                gameOver = true;
            }

            currentPiece = null;
        }
    }
}
```

dropPiece é responsável por iniciar uma nova peça no jogo, atribuindo-lhe uma posição inicial e adicionando-a à cena se nenhuma peça estiver ativa no momento. Ela seleciona uma peça aleatória, define sua posição vertical e a nomeia com base na quantidade de objetos na cena antes de adicioná-la. *updatePiecePosition* gerencia a movimentação da peça ativa, movendo-a verticalmente para baixo a cada chamada.

Figura 4. Trecho do código checa a colisão das peças.

```
function colidesWithSceneObjects(piece, excludedGroupsUuid = []) {
    // Check for collision with all other pieces in the scene
    for (let otherPiece of scene.children) {
        if (otherPiece !== piece && !excludedGroupsUuid.includes(otherPiece.uuid) && checkCollision(piece, otherPiece)) {
            // Collision detected, stop moving the current piece
            return true;
        }
    }

    return false;
}

function checkCollision(group1, group2) {
    for (let obj1 of group1.children) {
        const box1 = new THREE.Box3().setFromObject(obj1);

        for (let obj2 of group2.children) {
            const box2 = new THREE.Box3().setFromObject(obj2);

            if (box1.intersectsBox(box2)) {
                console.log("Collision detected between:\n\t",
                    group1.name, "\n\tand", group2.name);

                console.log("Intersection occurs at:", box1.intersect(box2));
                return true;
            }
        }
    }

    return false;
}
```

O código acima checa se a peça colidiu com qualquer outro objeto da cena, exceto os UUIDs que estão listados em *excludeGroupsUuid*. Se uma colisão for detectada, a função retorna *true*.*checkCollision* verifica se há colisão entre dois grupos. A função calcula a caixa delimitadora e verifica se essas caixas se intersectam.

5. Resultado Esperado

Ao final do trabalho, espera-se ter desenvolvido uma aplicação visualmente satisfatória e iterativa, que possa ser de fácil uso e manejo ao usuário. Espera-se que o jogo possa ter um funcionamento similar o Tetris, com sua modelagem 3D e jogabilidade. Desenvolvendo esse trabalho, tivemos um ampliamento sobre nosso conhecimento acerca da Computação, no geral, e sobre a Computação Gráfica. Pudemos desenvolver nossa lógica ao lidar com a criação de um jogo medianamente complexo.

5.1. Jogabilidade

Os pontos são contabilizados quando o jogador consegue montar um conjunto de blocos encaixados, sem espaço entre eles. O jogo é zerado ao atingir o topo da grade. O usuário pode girar e mover a peça pelo espaço da grade. Para movimentar a peça, o usuário deve usar as letras A, W, S e D e, ao pressionar SHIFT + A/W/S/D, girará a peça. Após a colisão com outra peça ou com a base do cenário, ela não irá mais se mover. A pontuação se dará quando o usuário conseguir completar uma linha de blocos interia, sem espaços e de uma margem a outra.

Referências

Manssour, I. H. Introdução à computação gráfica. *Faculdade de Informática, PUCRS, Av. Ipiranga 6681, Prédio 30*.

Santos, E. T. ComputaÇÃo grÁfica: Estado da arte e a pesquisa na usp. *USP*.

Three.js. Fundamentals. Accessed on May 10, 2024.