

# **Projeto Final de Graduação**

## **Realidade Aumentada: Uma Abordagem Utilizando Kinect**

Erick Luis Moraes de Sousa

## 1. Resumo

## 2. Introdução

O mercado de games é, atualmente, o ramo de entretenimento que mais gera receita, chegando a ultrapassar os lucros obtidos pela indústria cinematográfica e musical juntas. O consumo deste conteúdo ocorre nas mais diversas plataformas, como Consoles, PCs, Celulares e Tablets. Serviços de venda de games digitais, como a Steam[1] e a Origin[2], tornaram-se indispensáveis para este nicho de consumidores.

Em uma tentativa de continuar inovando, as empresas de vanguarda da atual geração de consoles (Sony, Nintendo e Microsoft) resolveram apostar em uma nova forma de entretenimento, que permitiria que o usuário interagisse com o sistema de maneira mais interativa. A saída encontrada estava em reformular o componente mais clássico dos consoles: O Controle.

Neste projeto, iremos nos restringir ao uso do periférico Kinect[3] da Microsoft para desenvolver um game de batalha 2D (com suporte para até 4 jogadores) utilizando Realidade Aumentada. Os participantes do jogo se enfrentam em uma arena customizável, criada a partir de simples objetos encontrados em casa, dispostos sobre uma superfície plana.

Durante a fase de implementação, utilizou-se a linguagem C#, a Framework XNA 4.0[4] e a biblioteca do Kinect em C#. Para este fim uma Game Engine foi implementada, com o intuito de proporcionar uma maior modularização e flexibilidade do projeto. A Seção 4.1 contém maiores detalhes sobre a organização estrutural da mesma.

## 3. Motivações

- Realidade Virtual é um ramo ainda pouco explorado, porém propício a inovação
- Baixo custo de desenvolvimento
- Problemas dinâmicos e em tempo real, os quais requerem soluções criativas e eficientes
- Lucrativo
- Aplicação e desenvolvimento de técnicas do estado da arte

## 4. Metodologia

A priori procurou-se levantar os principais problemas que poderiam ser encontrados durante o desenvolvimento. Podemos citar como exemplo a detecção dos objetos em si pelo Kinect e em como realizar a colisão entre um objeto físico e um objeto virtual. Após o levantamento, estudamos maneiras de solucionar os problemas encontrados tendo sempre em vista a necessidade de fluidez do jogo, portanto nossas abordagens deveriam ser computacionalmente baratas. Para garantir a execução uniforme do jogo em diversos hardwares, a atualização dos objetos do jogo foram implementadas com base no tempo decorrido desde a última renderização e não frame-a-frame. A primeira abordagem garante que tanto sistemas mais lentos como mais rápidos sejam executados de forma igual, pois o atraso é compensado em uma maior mudança, como por exemplo, no deslocamento de um objeto.

O próximo passo consistiu em estruturar uma Game Engine que fosse flexível o suficiente para suportar o grau de customização necessário por um jogo. Um modelo orientado a componentes[5] foi escolhido e as razões para tal escolha serão apresentadas na próxima seção.

Com a Game Engine criada, restavam apenas dois passos: o Scanner da Arena e a implementação de um módulo de física. Para o Scanner de Arena, testamos o uso do sensor de profundidade e a câmera de cor. No caso do sensor de profundidade, o ruído capturado pelo sensor durante a execução do projeto tornou sua utilização inviável, portanto restringiu-se o uso da câmera de cor. Tratou-se o problema utilizando algoritmos de limiarização. Dada a necessidade de manter um gameplay suave, optou-se por descartar técnicas de limiarização local, como o método de Niblack(1986) e Sauvola e Pitaksinen(2000).

#### 4.1. Game Engine

Apesar da enorme quantidade de Game Engines disponíveis[6], escolhemos por desenvolver a nossa própria utilizando a Framework XNA 4.0 como base, pois ela já provém diversas primitivas, como carregamento de texturas, renderização e um Game Loop (Figura 1) estável.

A estrutura da Game Engine está descrita na Figura 2. Os módulos estão descritos a seguir:

- **Renderer:** Este componente é responsável por criar uma interface entre o dispositivo de renderização e o mundo do jogo.
- **Asset Loader:** O Asset Loader é um wrapper em torno da classe de gerenciamento de conteúdo do XNA, porém de escopo global.
- **Keyboard Handler:** Detecta o estado atual das teclas, armazenando as informações em uma estrutura comum a todos os inputs.
- **GamePad Handler:** Possui o mesmo objetivo que o Keyboard Handler, porém relativo a controles de Xbox360.
- **Kinect Manager:** Contém a implementação de uma interface de comunicação entre o Kinect e o Game, bem como o processamento das imagens do Scanner.
- **World:** A classe World mantém uma referência para todos os objetos ativos do jogo. Para a implementação desta técnica, foi utilizado o algoritmo de Spatial 2D Hash[7], o qual subdivide o espaço do mundo em buckets de tamanho fixo. Objetos próximos são armazenados em buckets próximos. A Figura 3 exemplifica o particionamento de um espaço em grid utilizando o Spatial 2D Hash, onde os buckets 1, 2 e 7 contém, respectivamente, os conjuntos de bolinhas (vermelho), (vermelho, azul), (azul).

Todos os componentes da Engine foram implementados como Singleton[9], para garantir sua unicidade durante a execução e o acesso global a suas propriedades.

##### 4.1.1 Game Objects e Game Components

A principal forma de interação com o mundo do jogo se dá através de objetos da classe Game Object. Estes objetos têm como intuito envelopar funcionalidades, interagir com o mundo, realizar a troca de mensagens e tratar eventos. Afim de possibilitar uma maior flexibilidade, pode-se utilizar o operador de herança e estender diretamente esta classe.

Os Game Components implementam o comportamento em si que deseja-se atribuir a um determinado Game Object. A seguir descrevemos os principais comportamentos implementados:

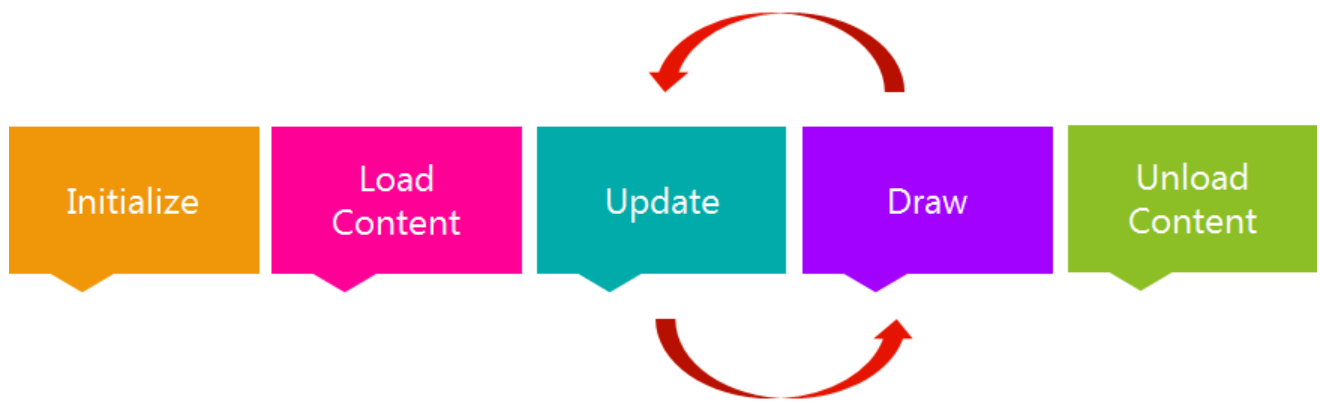


Figure 1. XNA Game Loop[8]

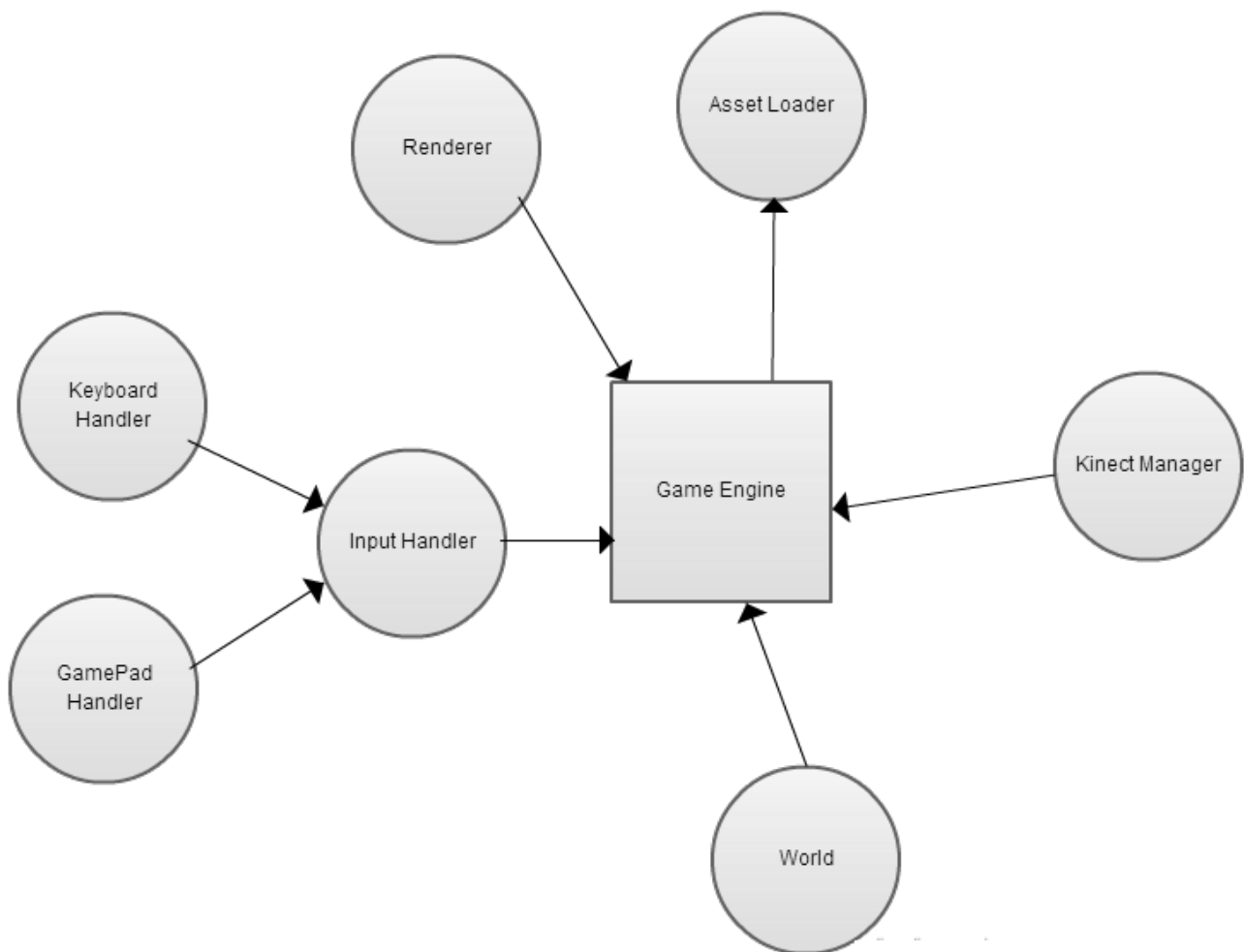


Figure 2. Arquitetura da Game Engine

- Transform: Responsável por armazenar a posição do jogador no mundo, sua rotação e escala.
- Render: Renderiza uma textura, aplicando operação de rotação, escala, transparência e a uma certa profundidade (coordenada Z do Transform, utilizada para definir a ordem de precedência

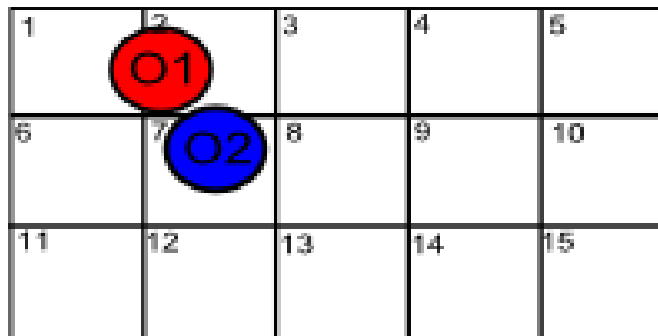


Figure 3. Partição espacial utilizando o algoritmo de Hash.

dos sprites).

- Physics: Define se um objeto ou não será colidível e com quem ele o será.

#### 4.2. Scanner de Arena

O propósito do Scanner de Arena é permitir que a mesma seja alterada dinamicamente durante o jogo, sem que isso proporcione atrasos (diminuição de frame rate). Usando o modo de câmera de cor, capturamos as imagens com resolução 640x480 a uma taxa de 30fps.

A técnica de limiarização global utilizada está descrita no algoritmo 1. Percorrendo a matriz linearizada de pixels, projetamos o pixel atual no espaço de cores de tons de cinza. Se este pixel estiver abaixo de um determinado Threshold (escolhido empiricamente com base na iluminação do ambiente, influência de sombras e o grau de destaque entre os objetos e o fundo - tipicamente branco), então marcamos ele como objeto (preto), caso contrário, fundo (branco). O resultado final é um vetor binário (0x0 ou 0xff).

**Data:** Imagem do Kinect linearizada, Threshold

**Result:** Vetor de objetos

```

for cada pixel  $p$  na imagem do
     $gs = (p.Red + p.Green + p.Blue) / 3$ 
    if  $gs \leq Threshold$  then
         $p = 0x0$ 
    else
         $p = 0xff$ 
    end
end

```

**Algorithm 1:** Algoritmo de Limiarização Global

Com apenas uma passada é possível separar os objetos do fundo.

## 5. Resultados e Conclusão

## 6. Projetos Futuros

## References

- [1] Valve Corporation. Steam. <http://store.steampowered.com/>. 2

- [2] Electronic Arts Inc. Origin. <https://www.origin.com/>. 2
- [3] Microsoft Corporation. Kinect for windows. <http://www.microsoft.com/en-us/kinectforwindows/>. 2
- [4] Microsoft Corporation. Xna 4.0. <http://msdn.microsoft.com/en-us/aa937791.aspx>. 2
- [5] Megan Fox. Game engines 101: The entity/component model. [http://www.gamasutra.com/blogs/MeganFox/20101208/6590/Game\\_Engines\\_101\\_The\\_EntityComponent\\_Model.php](http://www.gamasutra.com/blogs/MeganFox/20101208/6590/Game_Engines_101_The_EntityComponent_Model.php). 2
- [6] Jon Jordan. Engines of creation: An overview of game engines. [http://www.gamasutra.com/view/feature/132226/engines/\\_of/\\_creation\\_an\\_overview](http://www.gamasutra.com/view/feature/132226/engines/_of/_creation_an_overview). 3
- [7] Tristram MacDonald. Spatial hashing. [http://www.gamedev.net/page/resources/\\_/technical/game-programming/spatial-hashing-r2697](http://www.gamedev.net/page/resources/_/technical/game-programming/spatial-hashing-r2697). 3
- [8] Tess Fernandez. Bizzy bees step 1. <http://blogs.msdn.com/b/tess/archive/2012/03/02/bizzy-bees-step-1-setting-the-stage-xna-walkthrough.aspx>. 4
- [9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. Singleton. In *Design Patterns: Elements of Reusable Object-Oriented Software*, page 144, 1997. 3