Laboratório VISGRAF

Instituto de Matemática Pura e Aplicada

Estudo do Framework de Realidade Aumentada da Apple

Joao Pedro Xavier
Luiz Velho (supervisor)

Technical Report TR-18-05 Relatório Técnico

May - 2018 - Maio

The contents of this report are the sole responsibility of the authors.
O conteúdo do presente relatório é de única responsabilidade dos autores.

ESTUDO DA FRAMEWORK DE REALIDADE AUMENTADA (AR) DA APPLE - ARKIT

João P A Xavier
joaopedroaxavier@gmail.com

22 de maio de 2018

1 Introdução

A Realidade Aumentada (*Augmented Reality* ou simplesmente AR, em inglês) é a mistura da realidade com elementos virtuais. Com ela, têm-se a ilusão de que objetos renderizados em dispositivos estão de fato presentes no mundo real. Esse conceito tem sido bastante utilizado tanto para entretenimento como para simulações em geral.

Para alcançar esse objetivo, existe uma série de algoritmos e exigências de processamento que possibilitam que seja criada a ilusão de que o objeto está, de fato, presente no universo. É necessário que o dispositivo de inserção dos objetos seja capaz de detectar a posição e orientação das câmeras utilizadas para que os objetos sejam renderizados da forma correta. Além disso, é necessário uma boa detecção de luminosidade para que o objeto não esteja contrastando com a luz, o que atrapalha a imersão do objeto. Essas e outras preocupações são necessárias quando o conceito de Realidade Aumentada é trazido.

1.1 Motivações

A Realidade Aumentada (*Augmented Reality* ou simplesmente AR, em inglês) é um conceito não tão recente se compararmos com o surgimento de outros conceitos na computação, mas que, devido ao surgimento de novas tecnologias e o aumento na capacidade de processamento das máquinas atuais, tornou-se explorável. Recentemente, a Apple lançou uma framework para desenvolvimento de aplicativos AR, o ARKit, que permite que desenvolvedores *indie* ou grandes empresas possam criar aplicativos para entretenimento ou para dar suporte a outros projetos maiores.

Um grande exemplo a ser citado é o aplicativo desenvolvido pela IKEA (IKEA, 2017) para simulação dos móveis à venda no catálogo, de forma que o cliente pode verificar se um móvel vai agradá-lo quando for colocado em sua casa, como exemplificado na figura 1.

1.2 Embasamento Teórico

Para desenvolver uma aplicação em Realidade Aumentada, é necessário que, de alguma forma, o dispositivo utilizado tenha a capacidade de *reconhecer* a posição e a orientação dele com relação ao ambiente em tempo real. Esse processo é comumente chamado de *tracking*.

Normalmente, os dispositivos utilizam câmeras para capturar imagens do ambiente. A partir dessas imagens, é possível aplicar algoritmos que detectam pontos especiais, como por exemplo o SIFT (LOWE, 1999). Esse tipo de algoritmo é importante para especificar pontos de referência para a aplicação. Verifica-se o impacto desses pontos quando uma aplicação AR é feita em um ambiente rico em detalhes (como um local luminoso e com grande variedade de objetos e texturas), em

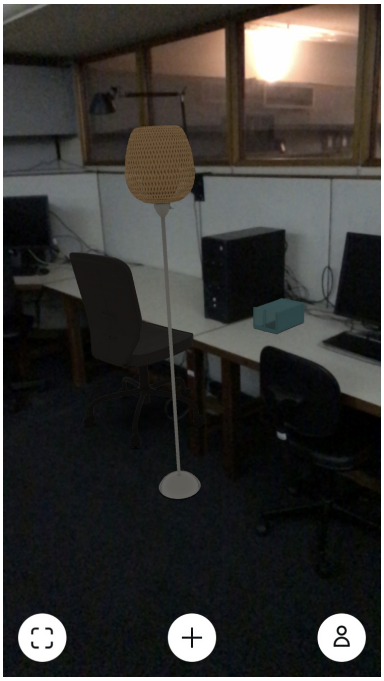


Figura 1: Imagem exemplificando a utilização do app da IKEA.

contraposição a um ambiente com poucos detalhes (como uma parede lisa ou um ambiente com pouca luminosidade). Após a captura e processamento das imagens, é feito um mapeamento do ambiente em que o dispositivo se encontra. O mapeamento é melhorado com uso de múltiplas câmeras, acelerômetros e outras ferramentas presentes em um *smartphone*, por exemplo, o que torna esse tipo de dispositivo bastante usado para aplicações AR.

1.2.1 Visual Inertial Odometry

O ARKit, em especial, utiliza a técnica de *Visual Inertial Odometry* (VIO) (NISTÉR; NARODITSKY; BERGEN, 2004) para associar as imagens retiradas pelas câmeras do aparelho Apple com sua posição e orientação. Essas imagens também são utilizadas para estimar o movimento (tanto em posição como em rotação) que está sendo realizado. Essa informação é importante pois é necessário saber onde e de que forma os objetos devem ser renderizados na tela.

O termo *Inertial* é dado devido à utilização de outros sensores do dispositivo (acelerômetros e giroscópios) para tornar o *tracking* mais preciso. Esses dispositivos foram aproveitados devido ao projeto *CoreMotion* da Apple. (DEVELOPER, 2014)

1.2.2 Scene Understanding

Um fato importante que foi lidado pelo ARKit é que apenas o posicionamento dos objetos não é suficiente para a imersáo destes no ambiente. É importante, também, identificar outros atributos como a luminosidade do local. Essa preocupação com atributos do ambiente é chamada de *Scene Understanding*.

Com o auxilio do sensor de luminosidade das câmeras, os objetos são renderizados com iluminação e sombreamento adaptados, atingindo o objetivo de impactar na imersão do objeto no ambiente.

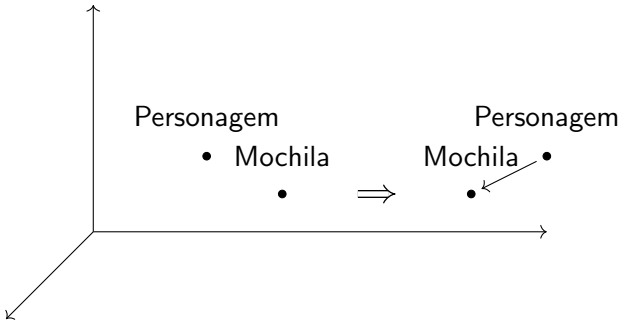
1.2.3 Grafo de Cena

O Grafo de Cena (FRENCH; ROBOTHAM, 2001) é uma maneira de estruturar hierarquicamente os elementos gráficos de uma cena. Essa estrutura permite utilizar posicionamento relativo entre objetos, o que muitas vezes é mais interessante que o posicionamento global do objeto.

A estrutura do Grafo de Cena é, essencialmente, uma árvore direcionada[1]. Cada vértice representa um objeto com sua posição na cena relativa ao

[1]Podemos definir uma Árvore Direcionada com um grafo direcionado e acíclico (DAG) com apenas um nó inalcançável, que é chamado de nó raiz.

(a) Esquema gráfico da cena



(b) Representação do Grafo

Figura 2: Esquema de um Grafo de Cena. É importante observar que o *transform* de alguns elementos pode ser alterado ao ser criada a relação no grafo.

vértice pai. As arestas do grafo representam a relação hierárquica de dois objetos (i.e. um está posicionado relativamente ao outro). Nas cenas gráficas, existe o nó raíz ou nó origem, que encarrega de atuar como a origem do sistema de coordenadas na cena.

Na implementação da *Apple* ele é referenciado diretamente como *rootNode*, dentro da classe *ARSCNView*. Essa funcionalidade não se dá diretamente pelo ARKit, mas sim pelas frameworks gráficas implementadas pela *Apple*, o SceneKit e o SpriteKit. Para provocar o posicionamento relativo do elemento B em relação ao elemento A, utiliza-se a chamada do método *addChildNode*, que adiciona o objeto B como filho do elemento A.

```
B.addChildNode(A);
```

Com o Grafo de Cena, é possível também realizar movimento relativo entre os objetos gráficos. É válido garantir que os *Transforms* dos objetos estejam alinhados corretamente, do contrário o movimento não será realizado como se deseja.

2 Realidade Aumentada (AR) com o ARKit

O ARKit, como já mencionado, é a ferramenta implementada pela Apple para desenvolver aplicações em Realidade Aumentada. A própria *framework* fica responsável por abstrair toda a parte do *tracking*, livrando o desenvolvedor desse tipo de preocupação. Para essa parte, o ARKit utiliza o método *Visual Inertial Odometry*. (DEVELOPER, 2018)

O processo de desenvolvimento de uma aplicação AR utilizando ARKit começa com uma instanciação de uma ARSession. Esse objeto encontra-se como membro de uma ARSCNView instanciada ou pode ser propriamente instanciado. Para dar início, cria-se uma instância de configuração do *World Tracking*, da classe ARWorldTrackingConfiguration, e assim chama-se o método run() da sessão, passando a configuração como parâmetro.

```
let configuration = ARWorldTrackingConfiguration()
sceneView.session.run(configuration)
```

O ARWorldTrackingConfiguration é o objeto que gerencia a parte de *tracking* junto à câmera do

dispositivo. Ele é responsável, além do próprio rastreamento em si, por detectar planos e iluminação de forma a tornar o objeto virtual o mais realista possível.

Uma das partes do processo de *tracking* consiste em detectar pontos característicos (em inglês *feature points*).

2.1 Hit Testing e Detecção de planos

O ARKit é capaz de detectar superfícies planas, indicando seus tamanhos e posições. Devido às limitações do *tracking*, esses planos são usualmente imprecisos no começo da experiência. Com o passar do tempo, os algoritmos de detecção se habituam ao ambiente e os planos tornam-se mais precisos. Assim, dois planos inicialmente distintos são unidos quando percebe-se que, na verdade, eles pertencem a uma única superfície.

Essa detecção de superfícies é feita pelos métodos de hit testing presentes na framework, oferecendo o conteúdo encontrado pelo tracking. Através desses métodos, os planos encontrados são mostrados como objetos da classe ARPlaneAnchor, ou seja, um ARAnchor que representa um plano. Esses objetos são atualizados em tempo real, e o desenvolvedor pode adicionar comportamentos extras na ocorrência de adições, remoções e atualizações por meio dos métodos didAdd, didRemove e didUpdate, respectivamente.

```
func renderer(_ renderer: SCNSceneRenderer,
 ↪  didAdd node: SCNNode, for anchor:
 ↪  ARAnchor) {
    guard let planeAnchor = anchor as?
     ↪  ARPlaneAnchor else { return }

    showMessage("Plane detected!")
    showPlayButton()
}
```

O exemplo acima mostra na tela que um plano foi detectado. Tais métodos podem ser úteis quando uma aplicação pressupõe a existência de planos, impedindo o início da experiência enquanto um plano não é percebido. É importante também mostrar que os planos são constantemente alterados, embora a referência deles seja mantida (dessa forma é possível fixar planos especiais durante a execução do aplicativo). Essas referências persistem até que o objeto seja realmente destruído, no caso do *merge* de superfícies, por exemplo.

2.2 Integração com Unity: Unity ARKit Plugin

O *Unity* adotou o ARKit por meio de um *plugin* para sua plataforma: O *Unity ARKit Plugin* (UNITY, 2018). Ao importá-lo para um projeto no Unity, algumas ferramentas são carregadas, bem como diversos exemplos são disponibilizados para utilização, o que torna fácil o aprendizado da ferramenta pelo desenvolvedor. Além disso, a integração com a modelagem de cenas do Unity torna o posicionamento dos objetos e o manuseio deles muito mais facilitado.

Além disso, devido ao fato do Unity ser um ambiente muito mais utilizado para desenvolvimento de jogos e aplicações gráficas que propriamente o Xcode, há uma série de outras ferramentas disponíveis, o que torna a experiência de desenvolver um aplicativo AR muito mais fluida.

Uma outra ferramenta criada para integração do ARKit com o Unity foi o ARKitRemote, que descarta a necessidade do *deploy* do aplicativo durante os testes, de forma similar ao "game view" do Unity. Para isso, é necessário construir o aplicativo no iPhone ou dispositivo Apple em que se deseja testar o aplicativo pelo menos uma vez, com um objeto do ARKitRemote inserido na cena. Após a construção, é possível conectar-se ao dispositivo

durante a *game view*. Essa ferramenta encurta o tempo de *debugging* do desenvolvedor, aumentando a produtividade.

3 Aplicação: AR Duck Hunt

3.1 O jogo

Duck Hunt (R&D4, 1985) foi um clássico dos jogos *old school*, inicialmente lançado no Japão para o console NES em 1984. Anos depois o jogo foi expandido para outras regiões (América em 1985, e Europa em 1987).

Originalmente, o jogo consiste em, com o auxílio de um dispositivo em forma de pistola com sensores de luminosidade, acertar uma série de patos que surgem na *TV*. O jogo é dividido em *rounds*, e cada *round* tem uma quantidade de patos a ser acertada. Há também um limite de tiros que podem ser feitos por *round*. O jogo segue num estilo *arcade*, podendo ser jogado até uma quantidade indefinida de *rounds* até que o jogo decrete *Game Over*.

Figura 3: Tela do jogo Duck Hunt pra NES.

3.2 Inserindo a Realidade Aumentada

Esse jogo tem um comportamento que facilita uma interação agradável em realidade aumentada. O design dos patos pode ser inserido ao mundo real com modelos 3D cartunizados, por exemplo. Por preferências pessoais, os desenhos se tornaram modelos 3D *low poly* que representam os objetos e personagens presentes no jogo, e ainda mantendo o aspecto *old school* do jogo. Esse modelo foi criado utilizando o *Blender*, software de *design* de modelos 3D .

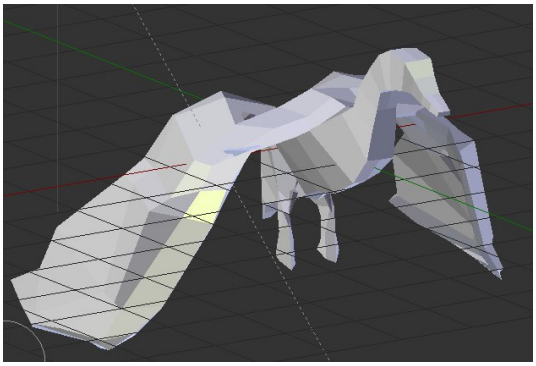


Figura 4: Modelo 3D de um pato utilizado no jogo.

Como o jogo consiste em mirar nos objetos que voam no universo do jogo, esse "universo" pode ser trazido ao mundo real. Como a interface de jogos *First Person Shooter* (FPS) é bem familiar a usuários que gostam de *games*, foi uma ideia interessante colocar esse tipo de interface com o usuário junto a essa experiência AR, criando uma visão similar ao que ele tem em jogos FPS.

A experiência de um jogo *arcade* também se encaixa quando o jogo é introduzido à Realidade Aumentada.

3.3 Implementação

O jogo foi feito utilizando Unity com o plugin para o ARKit, e foi testado em um iPhone 7.

A tela inicial é feita utilizando conceitos básicos de UI e Unity em geral, e explicar o passo a passo dessa parte foge o escopo dos detalhes técnicos aqui descritos. Após implementação do menu principal, o resultado é o mostrado na figura.

Figura 5: Menu principal do jogo.

Independente do aplicativo desenvolvido, é sugerido que o usuário mova a câmera ao seu redor por um tempo, de forma que o aplicativo possa ancorar-se e ter grande precisão nos algoritmos de *tracking*. No caso do Duck Hunt, esse problema do *tracking* inicial foi solucionado informando ao usuário que o jogo pode ser iniciado após uma superfície plana ser detectada. Essa abordagem foi feita, principalmente, por dois motivos:

1. Detectar uma superfície plana requer um estado de *tracking* satisfatório para a aplicação. Assim, quando o jogo for iniciado, não haverá erros de posicionamento dos objetos devido a um *tracking* fraco;

2. Fazer os patos surgirem de locais aleatórios podem ser um grande problema para a jogabilidade. Como a visão do jogador é restrita à tela do celular, seria necessário um conjunto de ferramentas extras que informassem a localização dos patos para que a experiência tornasse prazerosa. Com uma superfície plana fixa, o jogador sabe onde os patos podem surgir, o que contribui para uma jogabilidade mais amigável.

Com isso em mente, a implementação realizada foi pôr uma mensagem ao usuário para mover o celular pelo ambiente, com o objetivo de encontrar uma superfície plana. Essa mensagem é ilustrada pela figura.

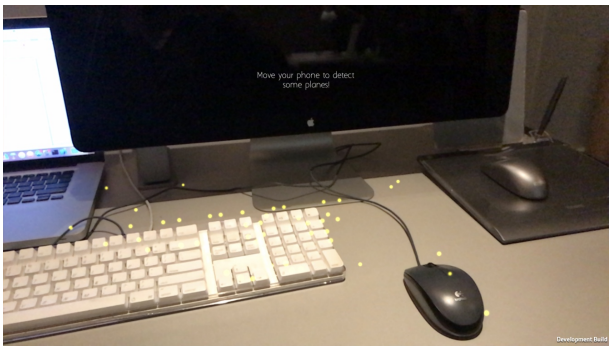


Figura 6: Tela mostrada ao usuário durante a fase de *tracking*.

Quando um plano é detectado, ele fica à mostra (destacado com uma borda azul) e é mostrada a mensagem de que o jogo está pronto para ser iniciado. Além disso, se o jogador não estiver satisfeito com a superfície, ele pode continuar movendo o celular para melhorar a superfície existente. Quando essa mensagem é mostrada, surge um botão para iniciar o jogo. A tela nessa situação está ilustrada pela figura.

O jogo é começado assim que o botão "I'm ready"é pressionado, dando início ao "round 1".

Figura 7: Tela mostrada ao usuário quando o jogo está pronto para ser iniciado. Observe o destaque em azul para a superfície plana detectada.

Sempre que há a transição de um round para o outro, é dado um tempo de trës segundos para que o jogador possa se reposicionar da forma que mais o agrada, além de dar uma separação bem definida entre os rounds.

O HUD do jogo é dividido em algumas componentes. São elas:

1. *Indicador dos rounds* (Canto superior direito da tela): Indica em qual *round* o jogador se encontra;

2. *Mira* (Centro da tela): Indica a posição e a orientação do tiro. Mais detalhes a seguir;

3. *Indicador dos patos* (Canto inferior esquerdo da tela): Indica a quantidade de patos do *round*, mostrando também quantos patos já foram acertados naquele *round*.

4. *Arma* (Canto inferior direito da tela): Essencialmente funciona como uma arma em um jogo *First Person Shooter*, com animações de *idle* (quando a arma fica ociosa, sem atividade) e *shoot* (quando um disparo é realizado). O modelo 3D da arma foi feito no Blender (REF), bem como o modelo do Pato. Os efeitos de fogo e ejeção dos cartuchos da munição são feitos com sistemas de partículas disponíveis gratuitamente na *Unity Asset Store*.

Os patos são gerados de acordo com o posicionamento do plano de referência. Essa estratégia, apesar de limitar a região de surgimento desses objetos, impede que o jogador não veja onde o pato foi gerado, o que causa uma impressão desagradável. Dessa forma, é designado, antes da instanciação do pato, uma posição com relação ao centro do plano. Quando o pato surge, ele realiza uma trajetória helicoidal (circular no plano XZ, uniforme no eixo Y).

Para acertar o pato, basta apontar a mira para a posição em que ele se encontra e tocar na tela. Com isso, a arma é disparada e é realizado um *raycast* para verificar a colisão com a cápsula de colisões presente no pato. Se sim, o objeto é destruído e essa informação é dada ao jogador. Do contrário, nada acontece. É importante ressaltar que o *raycast* é feito partindo a origem e orientado no eixo Z com relação à câmera, e não com relação à arma (ver seção 1.2.3), por motivos de agrado à jogabilidade.

A dificuldade do jogo é uma crescente com relação ao número de patos. Quanto maior o valor do *round*, maior a quantidade de patos por round. Uma restrição ao escopo do jogo, por enquanto, é que os patos necessitam de apenas um tiro para serem destruídos e não possuem propriedades especiais. A jogabilidade está ilustrada na figura.

4 Conclusão

Apesar da Realidade Aumentada ser um conceito não tão recente, havia diversas limitações tecnológicas que impediam seu desenvolvimento. O advento de técnicas mais recentes e integradas

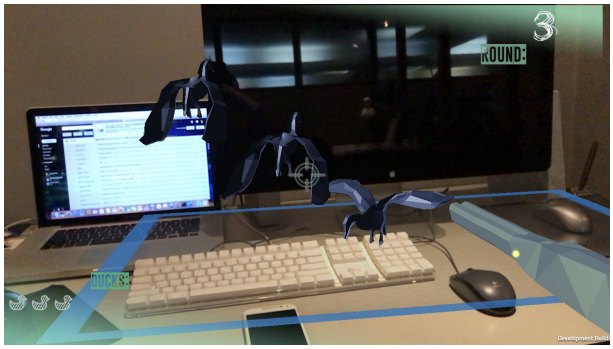


Figura 8: Tela de *gameplay* do jogo.

com *hardware* melhor preparado possibilitaram a introdução do AR em dispositivos comuns, como o iPhone. Além disso, a criação de uma *framework* para auxiliar desenvolvedores e empresas provocou uma maior aproximação entre esse conceito e o mundo cotidiano. Atualmente, as previsões de mercado para a Realidade Aumentada são bastante prósperas, e espera-se que mais ferramentas como o ARKit surjam e se aproximem cada vez mais do nosso dia-a-dia.

Referências

DEVELOPER, A. *Core Motion Framework Reference*. 2014.

DEVELOPER, A. *ARKit Framework Reference*. 2018.

FRENCH, M. T.; ROBOTHAM, J. S. *Time inheritance scene graph for representation of media content*. [S.l.]: Google Patents, 2001. US Patent 6,266,053.

IKEA. *IKEA Place*. 2017. Disponível em: ⟨https://play.google.com/store/apps/details?id=com.ikea.catalogue.android⟩ e ⟨https://www.ikea.com/gb/en/customer-service/ikea-apps/⟩.

LOWE, D. G. Object recognition from local scale-invariant features. In: IEEE. *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*. [S.l.], 1999. v. 2, p. 1150–1157.

NISTÉR, D.; NARODITSKY, O.; BERGEN, J. Visual odometry. In: IEEE. *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*. [S.l.], 2004. v. 1, p. I–I.

R&D4, N. *Duck Hunt, Nintendo Entertainment System*. [S.l.]: Nintendo, 1985.

UNITY. *Unity ARKit Plugin*. 2018. Disponível em: ⟨https://blogs.unity3d.com/pt/2018/02/16/developing-for-arkit-1-5-update-using-unity-arkit-plugin/⟩ e disponível para *download* em ⟨https://assetstore.unity.com/packages/essentials/tutorial-projects/unity-arkit-plugin-92515⟩.