MJ3D – Biblioteca de Algoritmos de Portais para a plataforma iOS

MATEUS JUNIOR CASSANIGA

ORIENTADOR: DALTON SOLANO DOS REIS

FURB – Universidade Regional de Blumenau
DSC - Departamento de Sistemas e Computação
Grupo de Pesquisa em Computação Gráfica, Processamento de Imagens e Entretenimento Digital
www.inf.furb.br/gcg

Roteiro

- Introdução
- Objetivos
- Fundamentação teórica
- Desenvolvimento
- Resultados e discussões
- Conclusão
- Extensões

Introdução

- Dispositivos móveis
  - Categoria de entretenimento
  - Evolução dos dispositivos
  - Jogos com gráficos e cenas complexos
  - Problema está no custo de processamento
- Técnicas de *culling*
  - Algoritmo de Portais
- Extensão do trabalho desenvolvido por Pandini (2012)

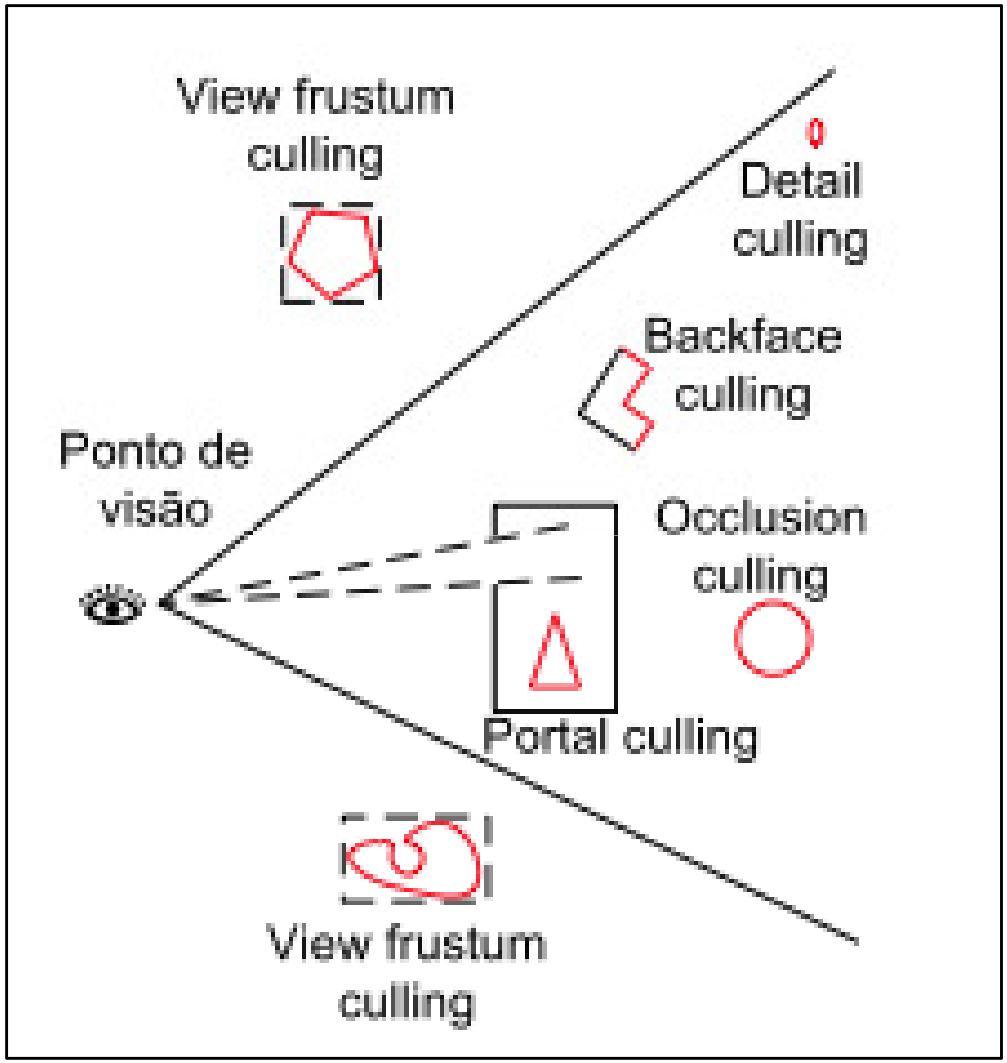
Objetivos

- Converter as funcionalidades já existentes no trabalho de Pandini (2012)
- Permitir utilizar cenários em 3D, mantendo os testes dos portais em 2D
  - Chamamos de 2D ½
- Testar outras formas para melhorar o desempenho do teste de visibilidade

Fundamentação teórica

Algoritmos de visibilidade

- Técnicas de *culling*
  - Determinar a visibilidade antes de enviar para a placa gráfica



Fonte: Carlos (2009, p. 19).

Fundamentação teórica

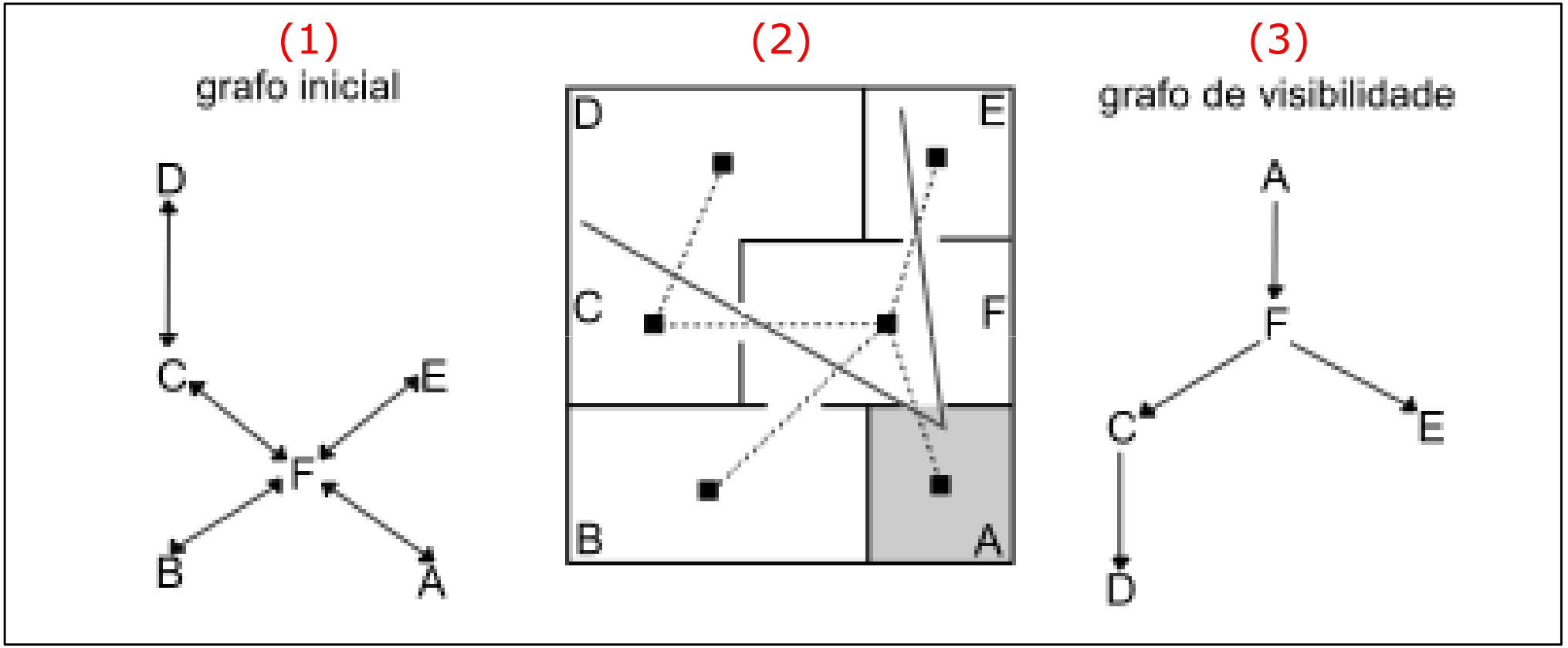
Algoritmos de Portais

- Ambientes internos
- Divisão do cenário
  - Salas
  - Portais
- Renderização de cenas em duas etapas
  - Primeira etapa
    - Onde a câmera se encontra
    - Novo *frustum* para cada portal encontrado
  - Segunda etapa
    - Quais objetos estão dentro do campo de visão

Fundamentação teórica

Algoritmos de Portais

- Grafos e ambiente

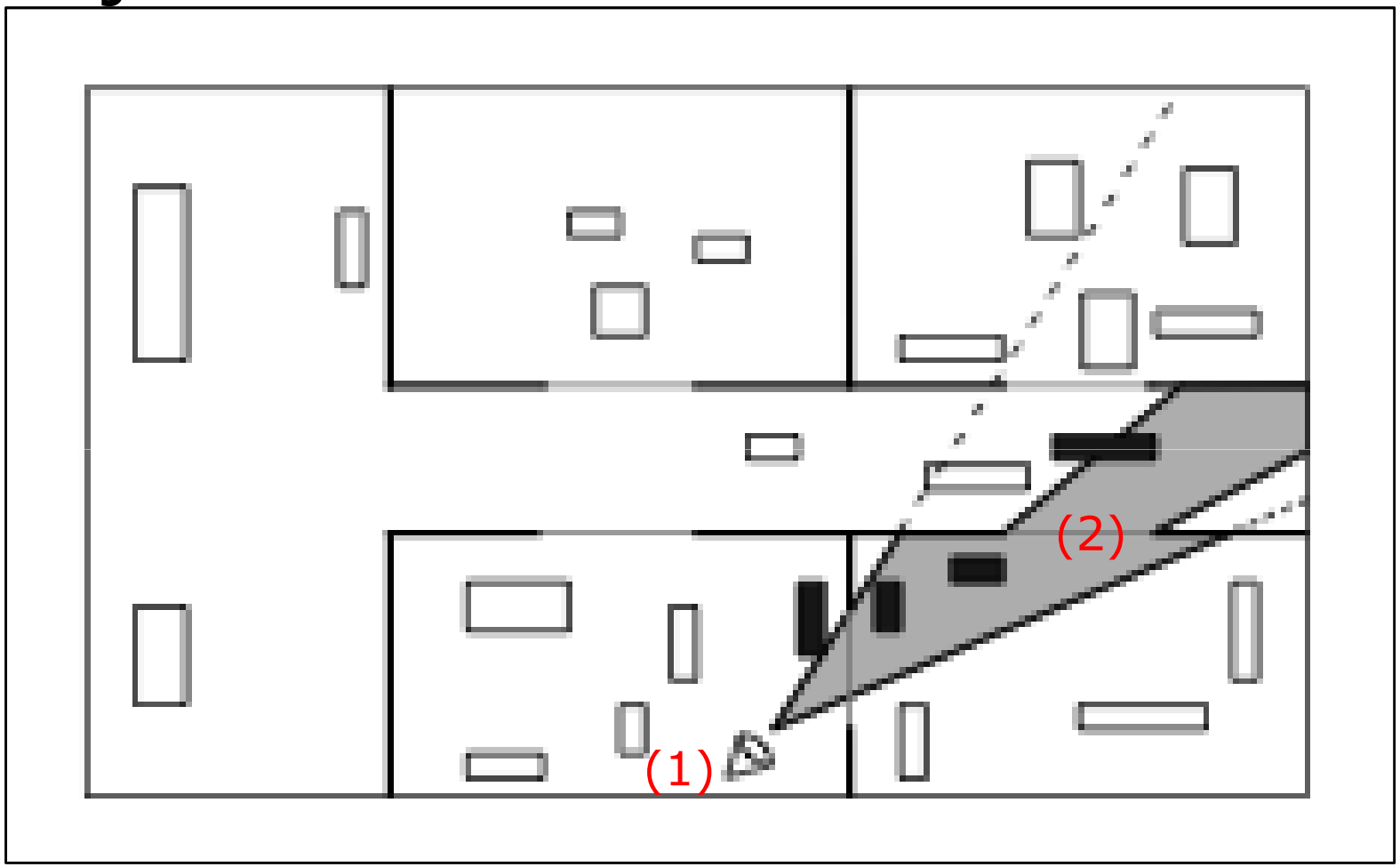


Fonte: Carlos (2009, p. 19).

Fundamentação teórica

Algoritmos de Portais

- Objetos renderizados



Fonte: Silva et al. (2003).

Fundamentação teórica

Plataforma iOS

- iPhone, iPod e iPad
- SDK
- Ambiente de desenvolvimento
- iOS Simulator

Fundamentação teórica

OpenGL ES

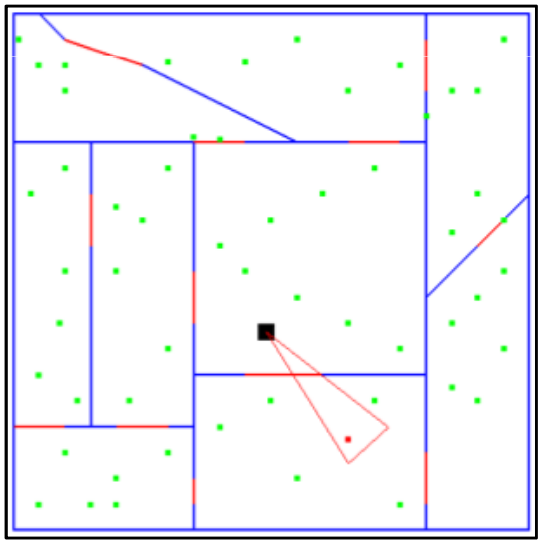
- Versão compacta

- Versão 2.0
  - iPad
  - iPhone 3GS ou superior
  - iPod *touch* de 3ª geração ou superior

Fundamentação teórica

Biblioteca de Algoritmos de Portais para a plataforma Android

- Desenvolvido por Pandini (2012)
  - Dispositivos móveis com Android
  - Bom desempenho
  - Sugestões
    - Configurar ângulo e tamanho do campo de visão
    - Editor de ambientes
    - Outras técnicas de visibilidade



Fundamentação teórica

Trabalhos correlatos

- Frustum Culling híbrido em CPU e GPU para visualização de modelos massivos em tempo real
  - Comparação entre algoritmo híbrido e comum
  - Algoritmo híbrido obteve melhor desempenho
- Renderização interativa em dispositivos móveis utilizando algoritmos de visibilidade e estruturas de particionamento espacial
  - Implementação e combinação de vários algoritmos
  - Renderização em tempo real pode ser obtida com sucesso
- Experiência de portais em ambientes arquitetônicos virtuais
  - Algoritmos de Portais
  - Em alguns casos o desempenho foi equivalente

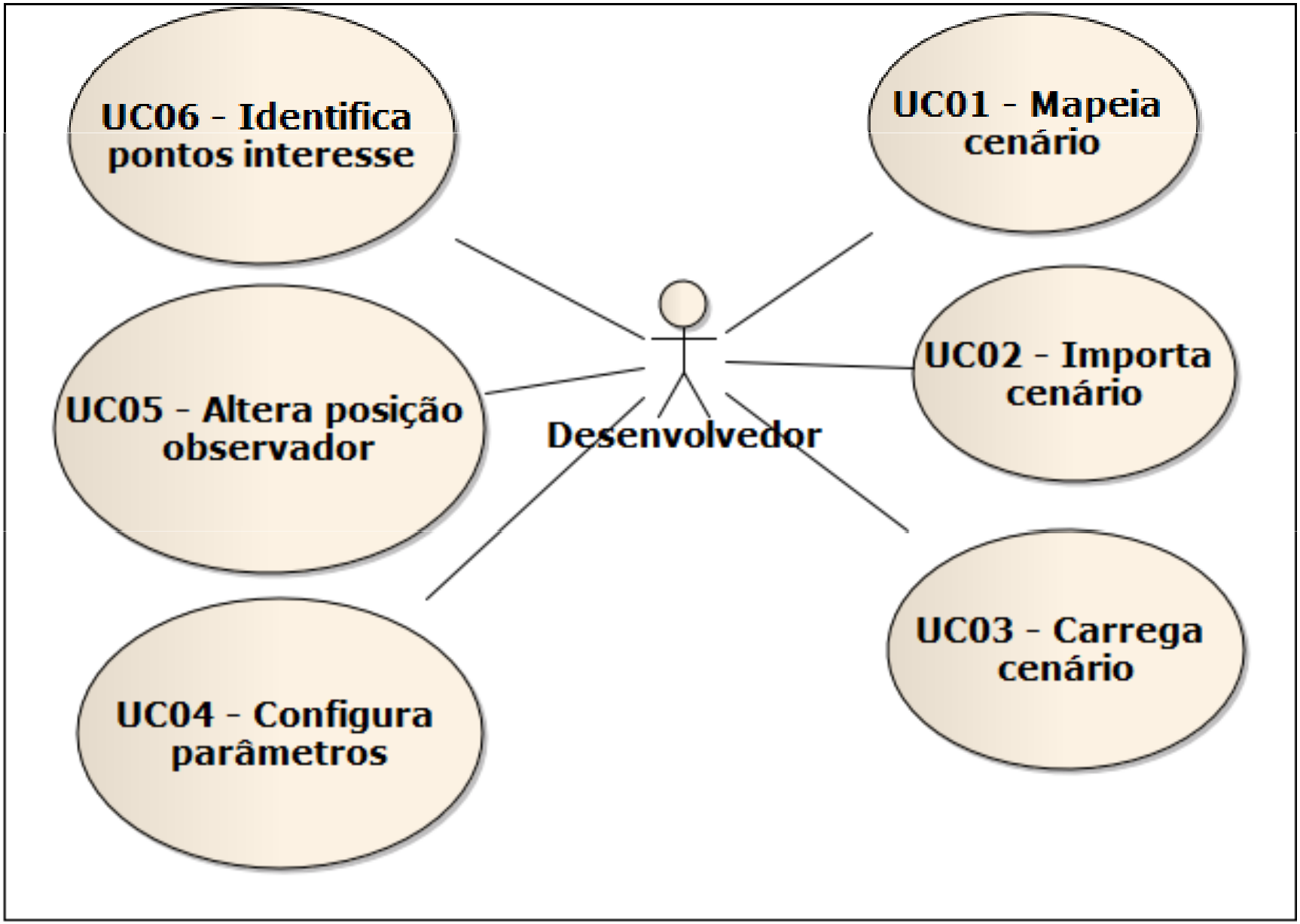
Desenvolvimento

Requisitos funcionais

- Ter uma implementação de Algoritmos de Portais
- Permitir que o ambiente seja particionado em salas
- Permitir a utilização de cenários 3D

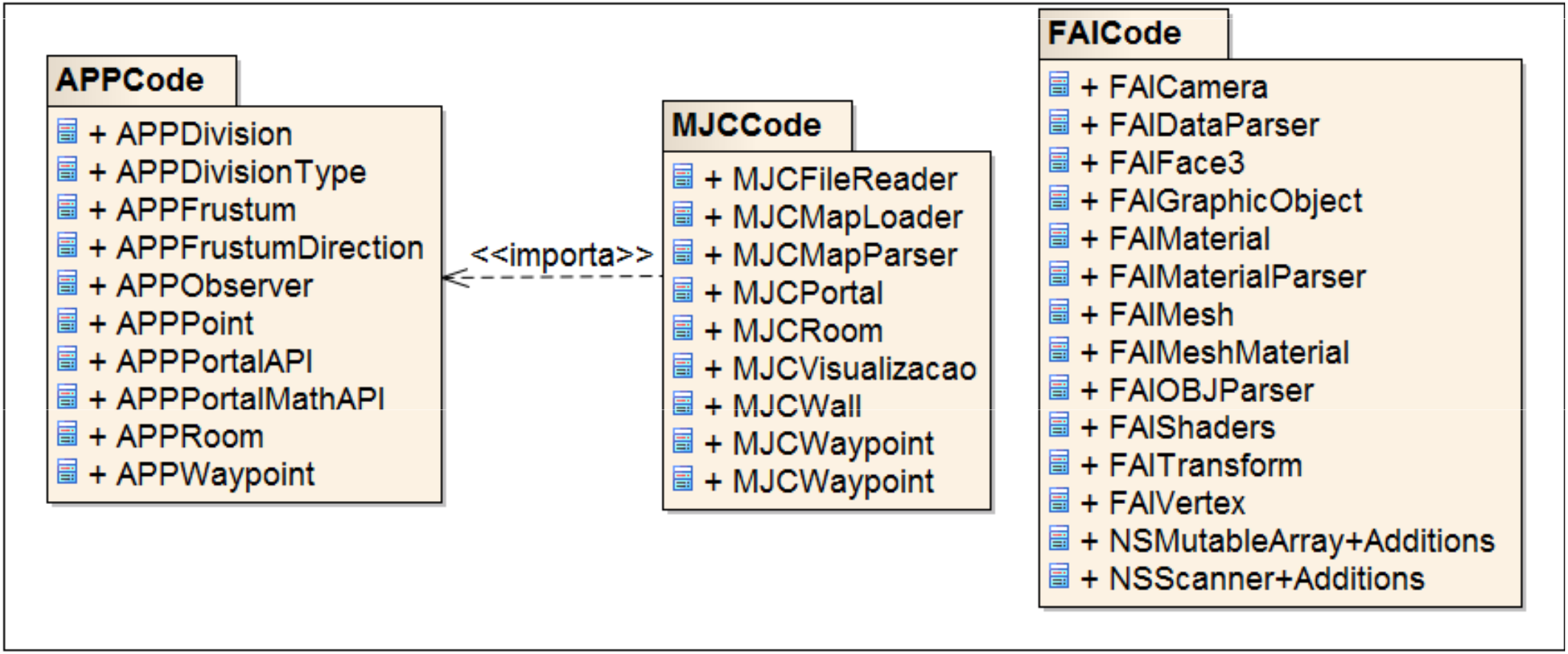
Desenvolvimento

Casos de uso



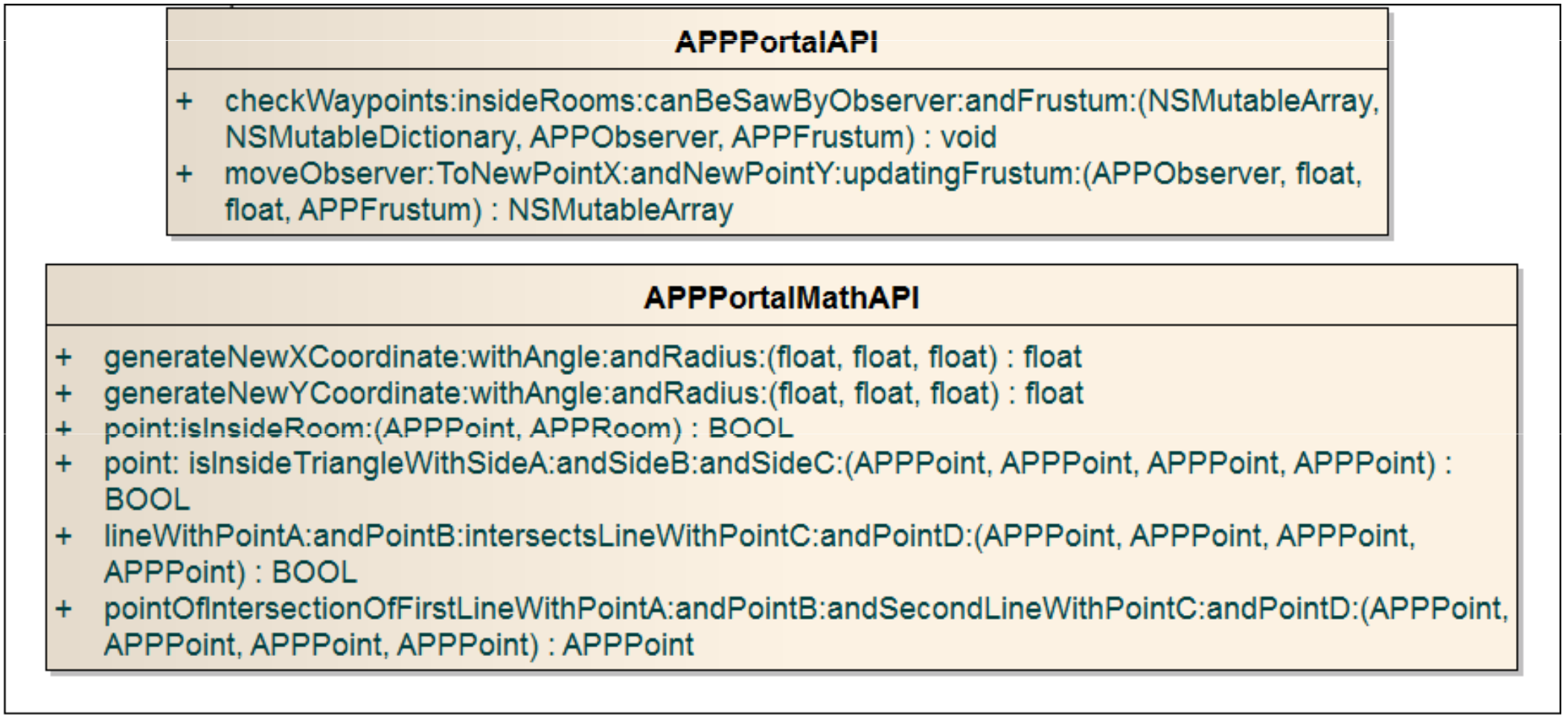
Desenvolvimento

Diagrama de classes: Pacotes



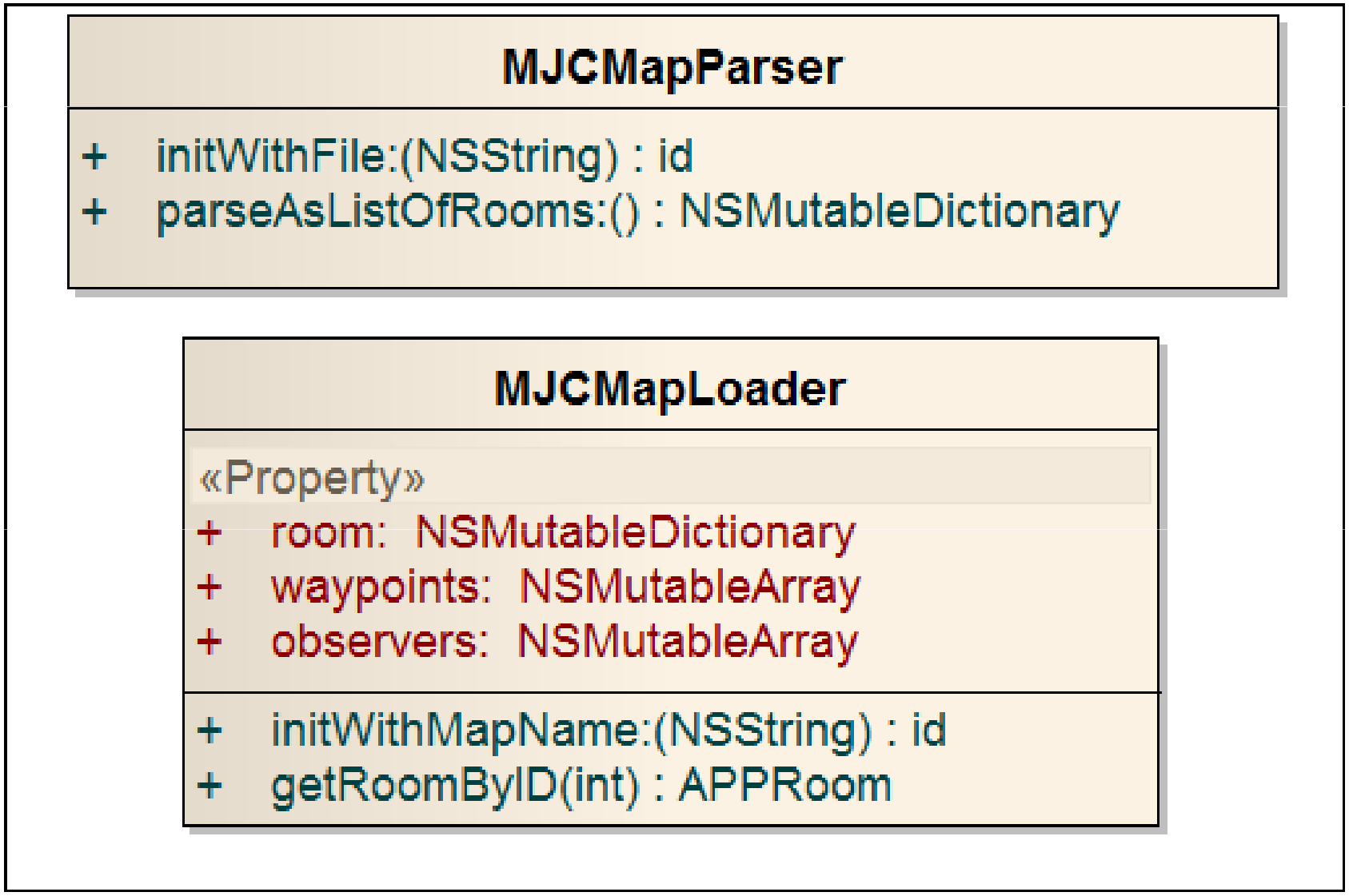
Desenvolvimento

Principais classes: APPCode



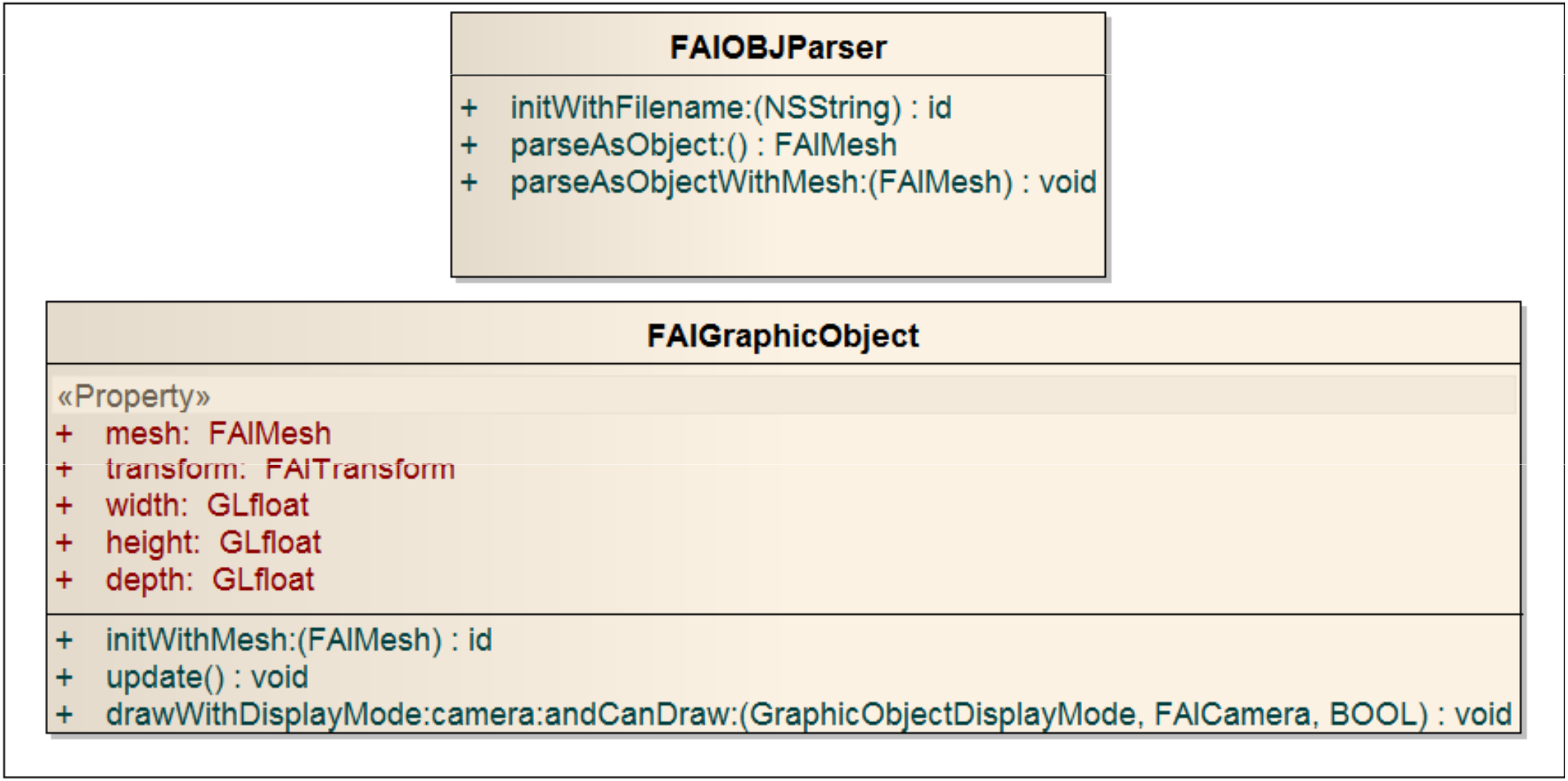
Desenvolvimento

Principais classes: MJCCode



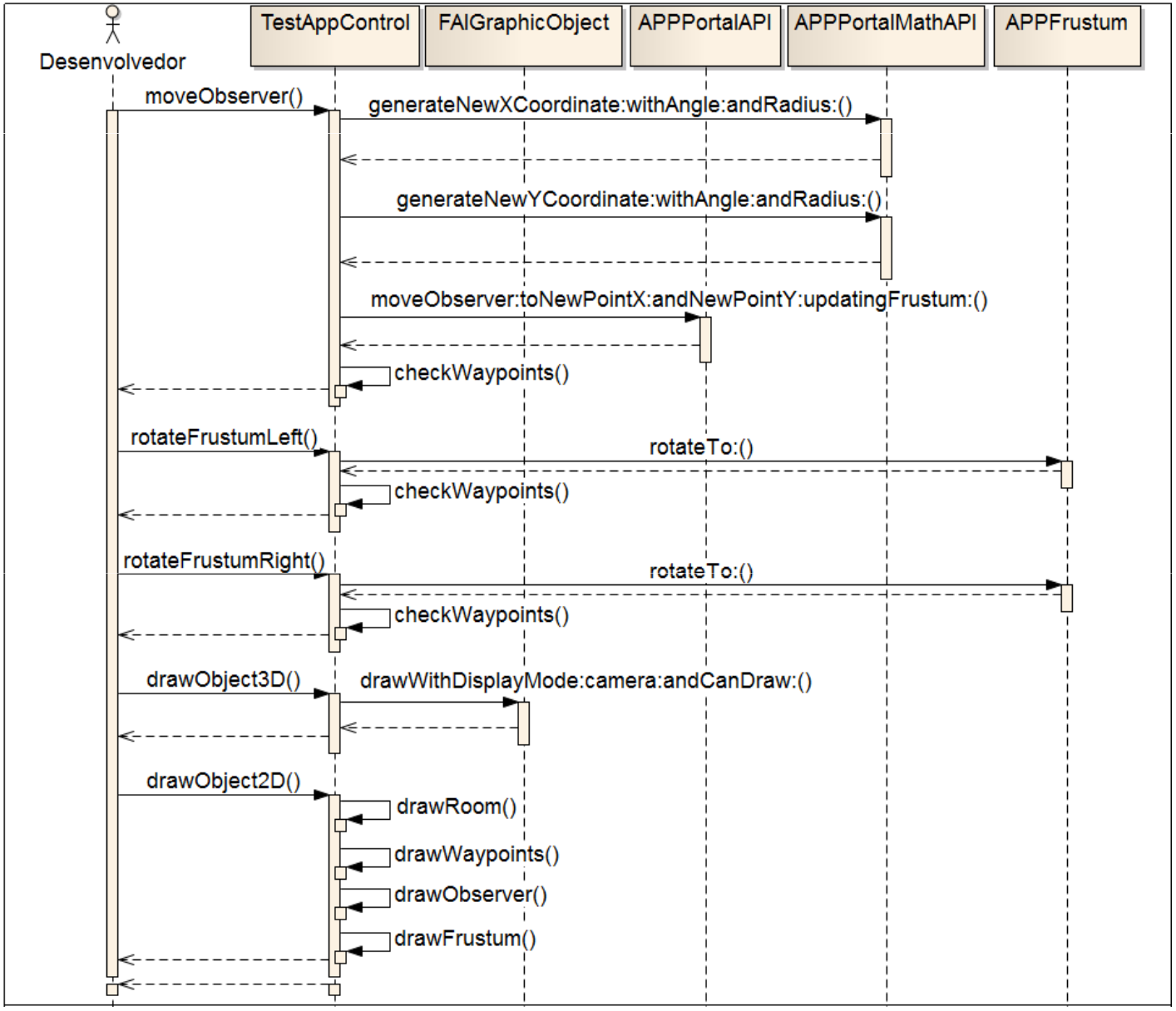
Desenvolvimento

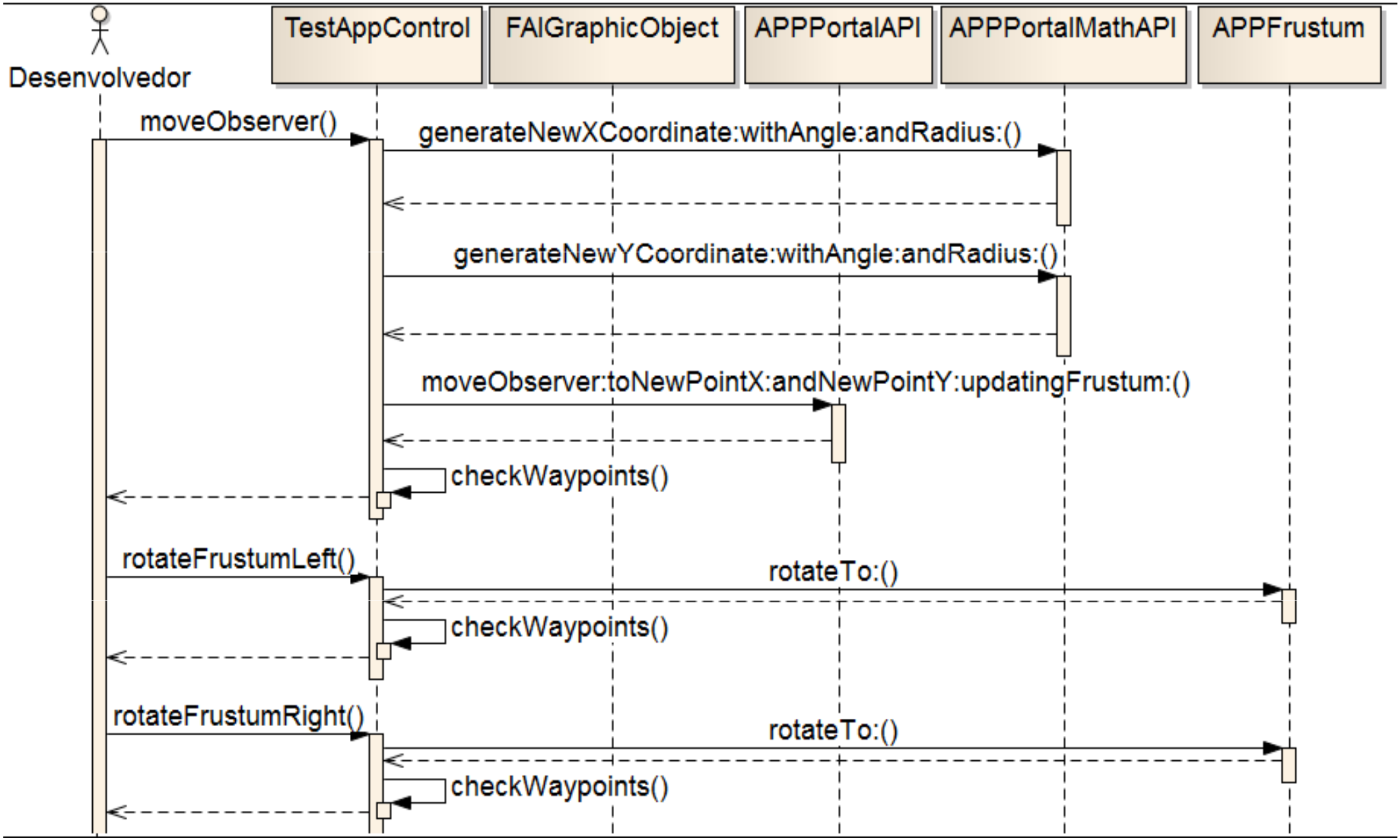
Principais classes: FAICode – Imianowsky (2013)



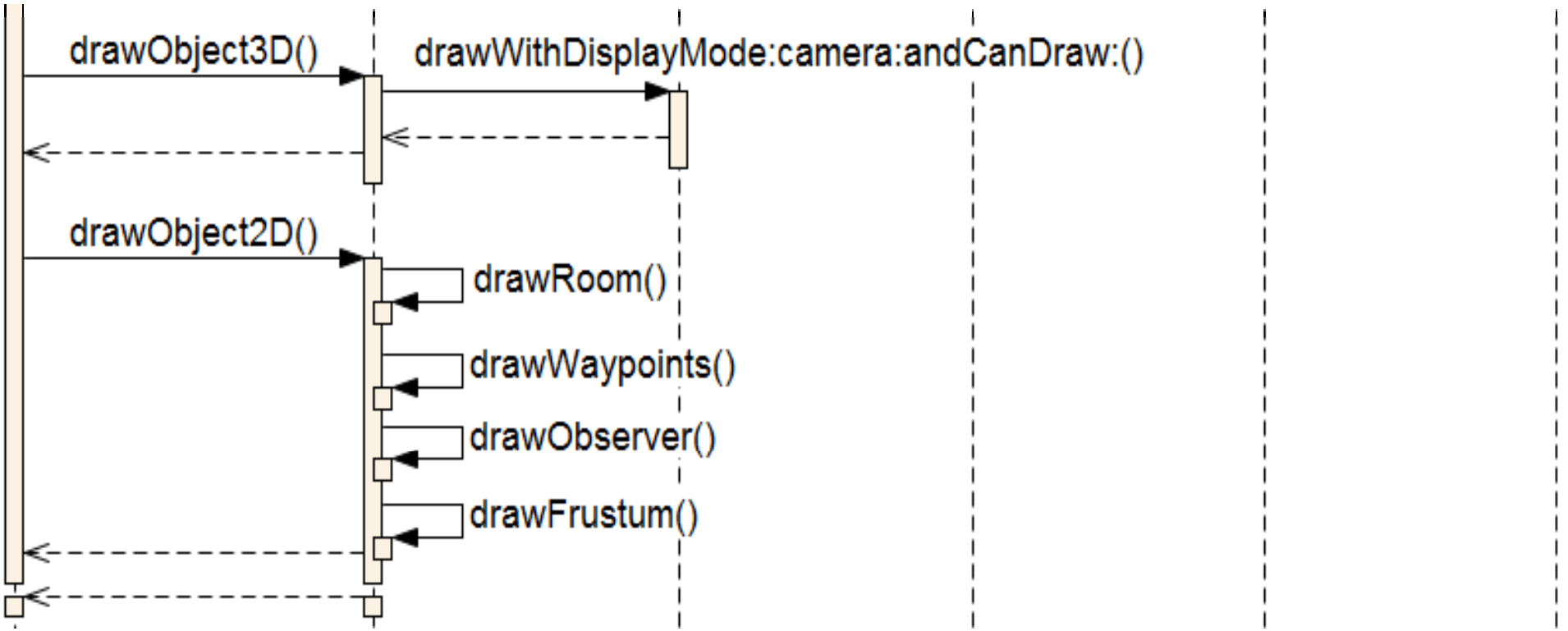
Desenvolvimento

Diagrama de sequência (ação de mover o observador no mapa)




Desenvolvedor
TestAppControl
FAIGraphicObject
APPPortalAPI
APPPortalMathAPI
APPFrustum
moveObserver()
generateNewXCoordinate:withAngle:andRadius:()
generateNewYCoordinate:withAngle:andRadius:()
moveObserver:toNewPointX:andNewPointY:updatingFrustum:()
checkWaypoints()
rotateFrustumLeft()
rotateTo:()
checkWaypoints()
rotateFrustumRight()
rotateTo:()
checkWaypoints()

Continuação...



Implementação

- Orientação a objetos
- Python
  - Add-on desenvolvido para o Blender
- Objective-C
- iOBJ (FAICode)
  - Desenvolvida por Imianowsky (2013)
- OpenGL ES 2.0
- XCode

Implementação

Add-on

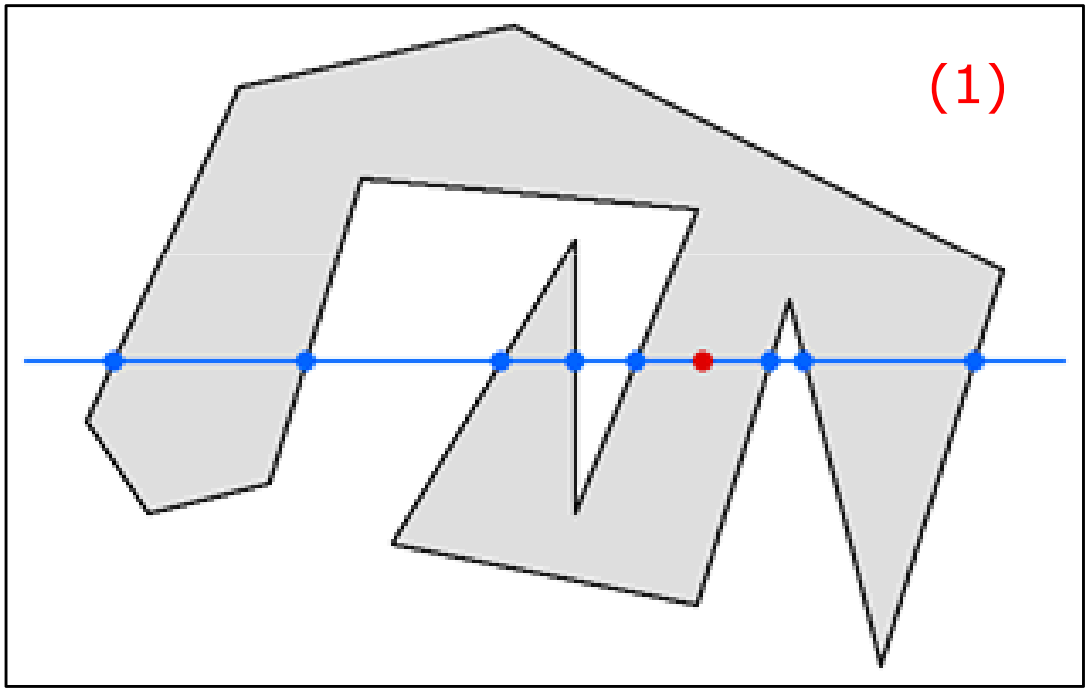
- Blender permite gerar OBJ e MTL
- *Add-on* acrescenta arquivo MAP
  - Informações do mapa utilizados pelo Algoritmo de Portais

```
o MAPA_SALA_01
e -0.000000 1.874998 0.050000 -0.000001 0.749999 0.050000
e -0.000001 0.749999 0.050000 -1.500000 0.749999 0.050000
e -5.624997 0.750001 0.050000 -3.749999 0.750000 0.050000   (1)
e -5.624997 5.624997 0.050000 -5.624997 0.750001 0.050000
e 0.000001 5.624996 0.050000 0.000001 4.499997 0.050000
e -5.624997 5.624997 0.050000 0.000001 5.624996 0.050000

o MAPA_SALA_02
e 0.375001 4.499997 0.050000 0.000001 4.499997 0.050000
e -0.000000 1.874998 0.050000 0.375000 1.874998 0.050000   (2)

o PORTAL_01_01_02
e -3.749999 0.750000 0.050000 -1.500000 0.749999 0.050000  (3)

o WAYPOINT_01_01
v -1.498443 3.765211 0.200000  (4)

o OBSERVADOR_01_01
v -3.015483 3.765211 0.200000  (5)
```

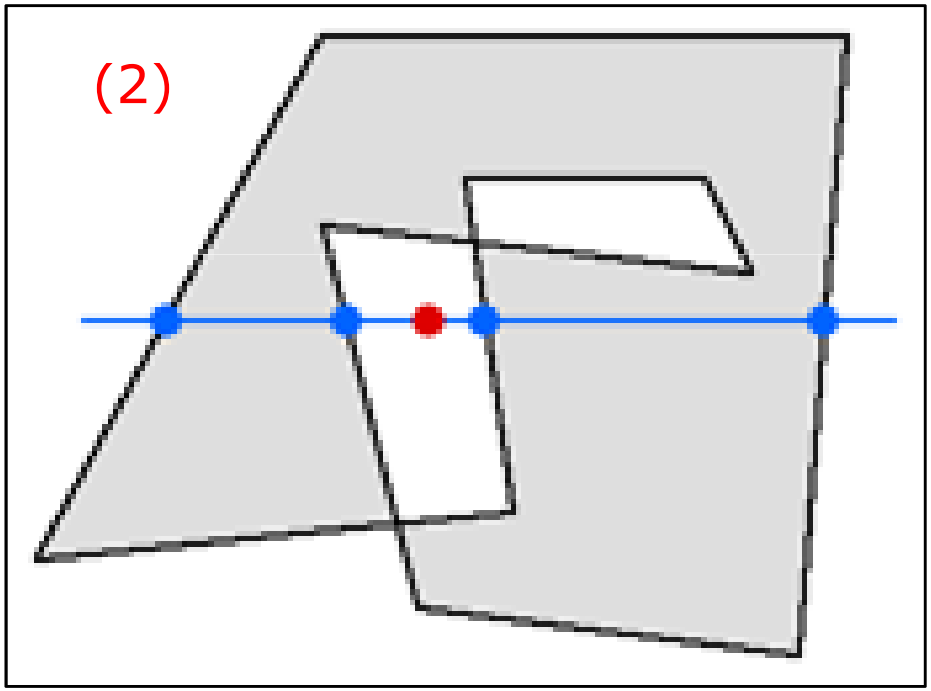
Implementação

Ponto dentro do polígono (FINLEY, 2007)

- Utilizado para verificar se o observador não está saindo da sala
  - Cria uma linha imaginária horizontal sobre o ponto testado
  - Quantidade de vezes que essa linha atravessa uma aresta do polígono



Fonte: Finley (2007).

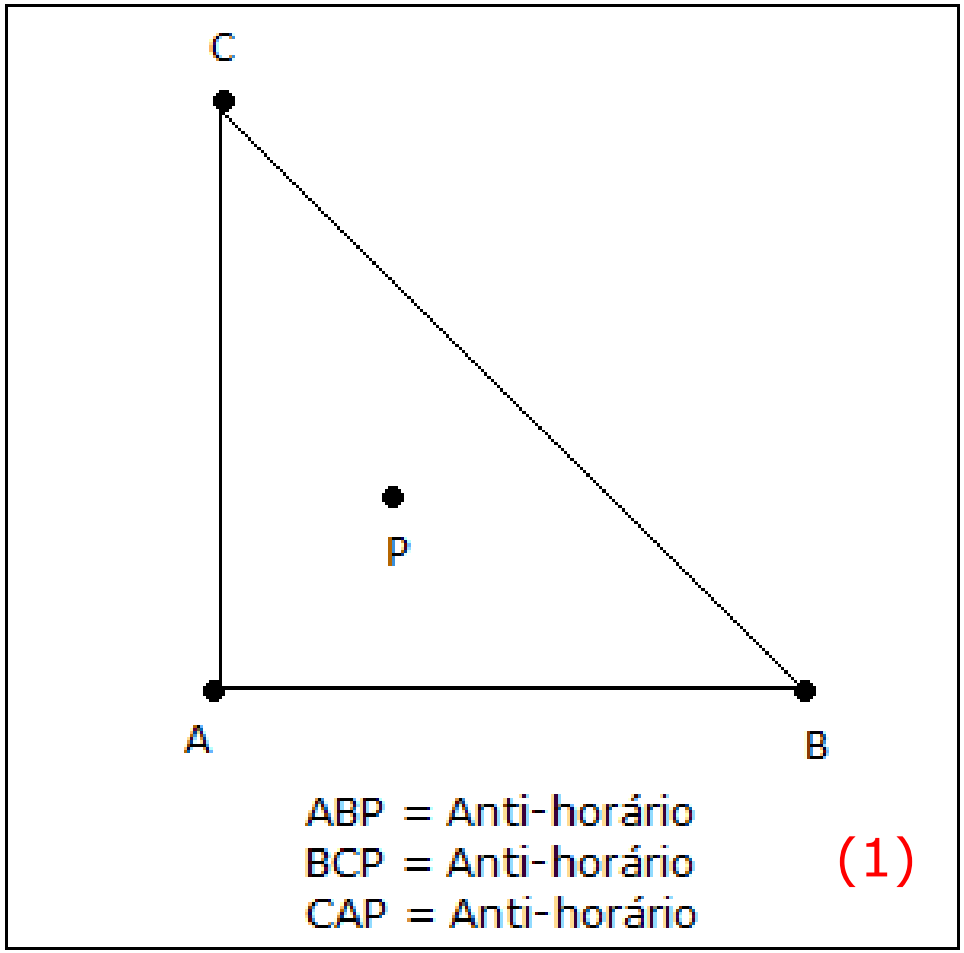


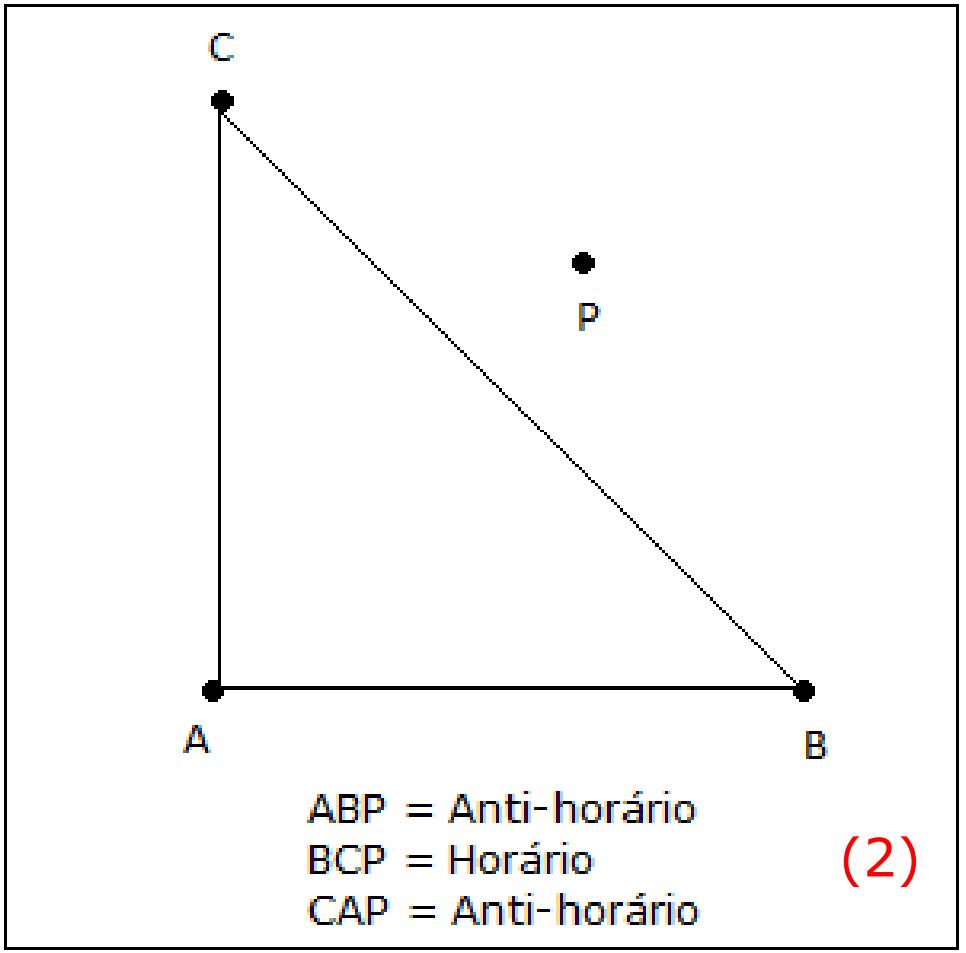
Fonte: Finley (2007).

Implementação

Ponto dentro do triângulo (BLACKPAWN , 2011; BOE, 2006)

- Utilizado para verificar quais pontos de interesse estão dentro do campo de visão do observador

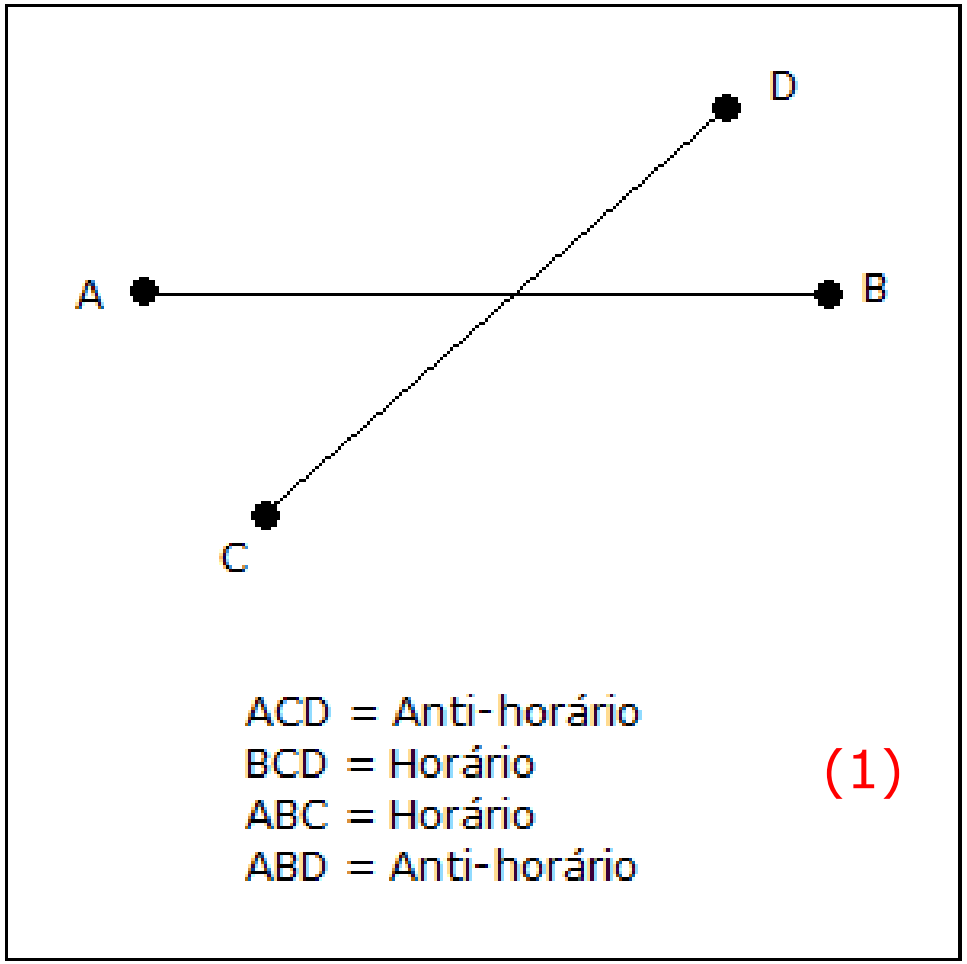


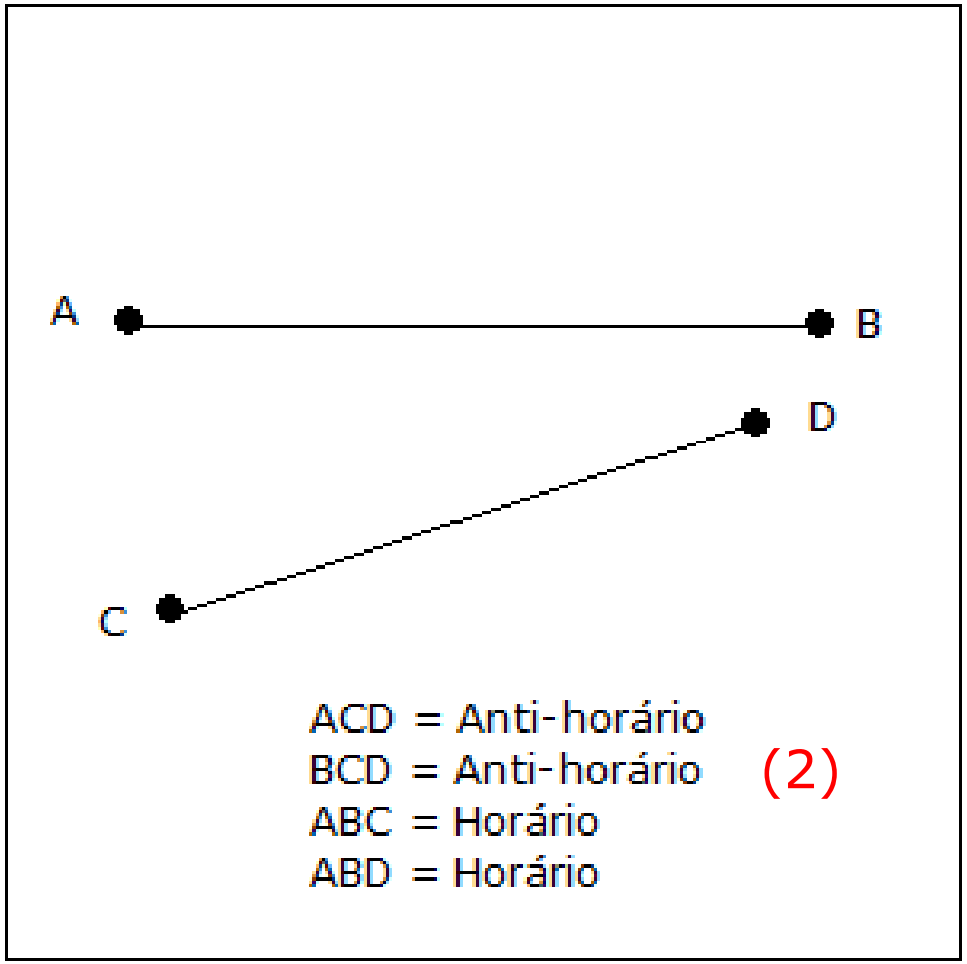


Implementação

Interseção de duas retas (BOE, 2006)

- Utilizado para verificar se não existe algo que impeça a visualização do ponto de interesse

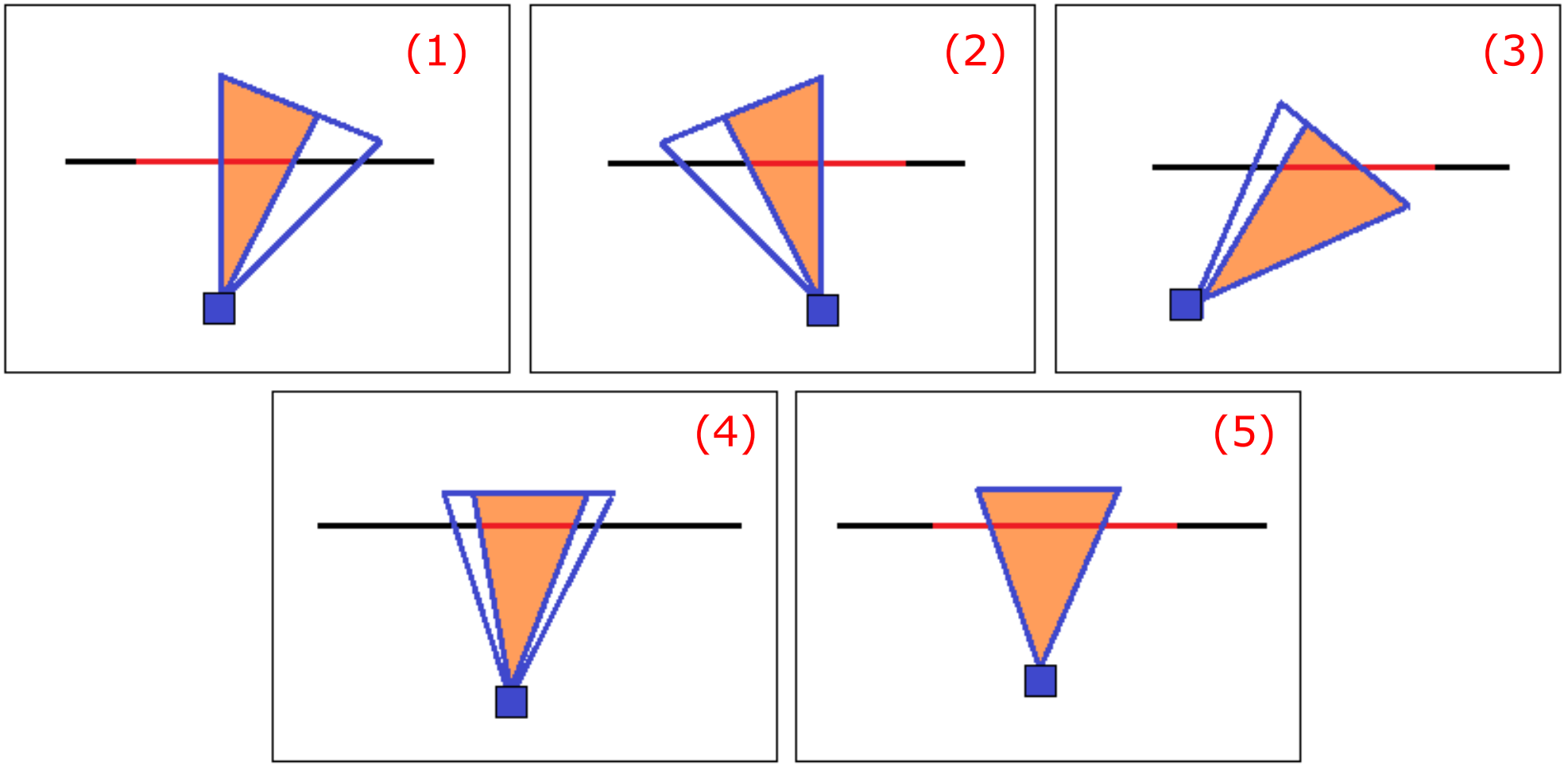




Implementação

Testes do *frustum* - Situações

- Situações em que é criado um campo de visão (*frustum*) auxiliar



Resultados e discussões

	Biblioteca desenvolvida	PANDINI (2012)	CARLOS; RAPOSO; SOARES (2010)	SILVA (2009)	SILVA et al. (2003)
Aceita modelos 3D	✓	⊘	✓	✓	✓
Importação de modelos	✓	⊘	✓	✓	✓
Executa em dispositivo móvel	✓	✓	⊘	✓	⊘
Algoritmo híbrido (GPU e CPU)	⊘	⊘	✓	⊘	⊘
Controle manual do observador	✓	⊘	⊘	✓	✓
Plataforma iOS	✓	⊘	⊘	⊘	⊘
Testes em 3D	⊘	⊘	✓	✓	✓

Resultados e discussões

Testes

- Dois tipos de testes
  - **Desempenho (em MS) dos métodos:**
    - Visão do observador
    - Mover observador
  - **Comparação de desempenho (em MS) do método responsável pela visão do observador**
    - Método convertido de Pandini (2012)
      - Coordenadas baricêntricas
    - Método desenvolvido neste trabalho

Resultados e discussões

Testes

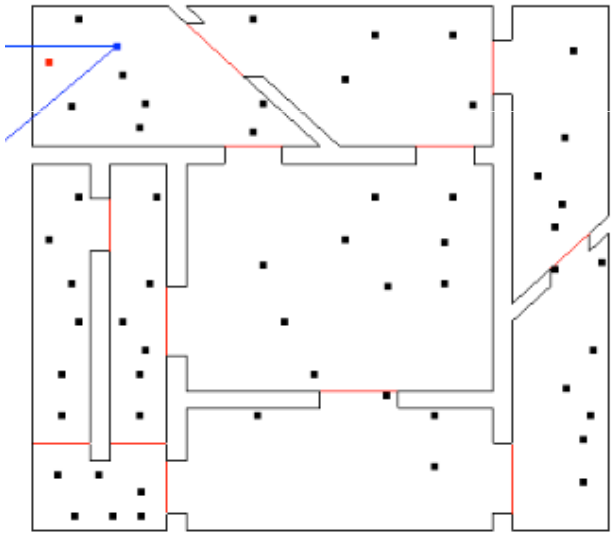
- Testes realizados em dois dispositivos
  - iPhone 4S
    - Processador Cortex-A9 *Dual-core* de 1GHz
    - 512MB de memória RAM
    - iOS 6.1.3
  - iPad 1ª geração
    - Processador Cortex-A8 de 1GHz
    - 256MB de memória RAM
    - iOS 5.1.1

Resultados e discussões

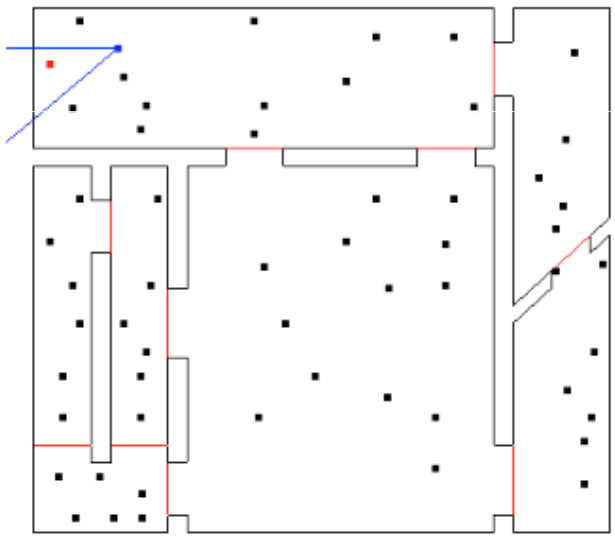
Testes

9 salas
12 portais

MAPA 1



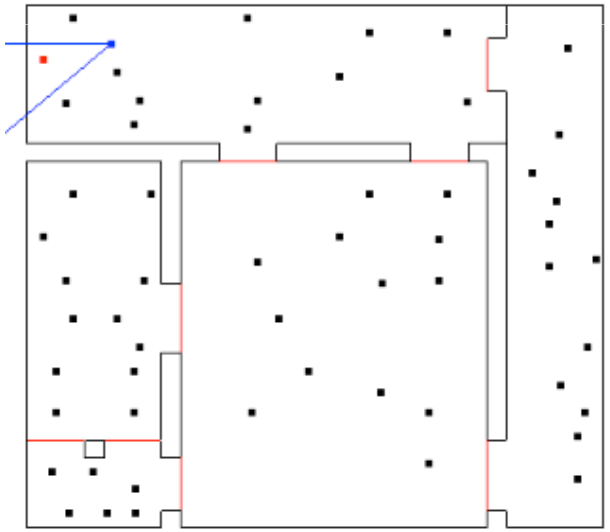
MAPA 2



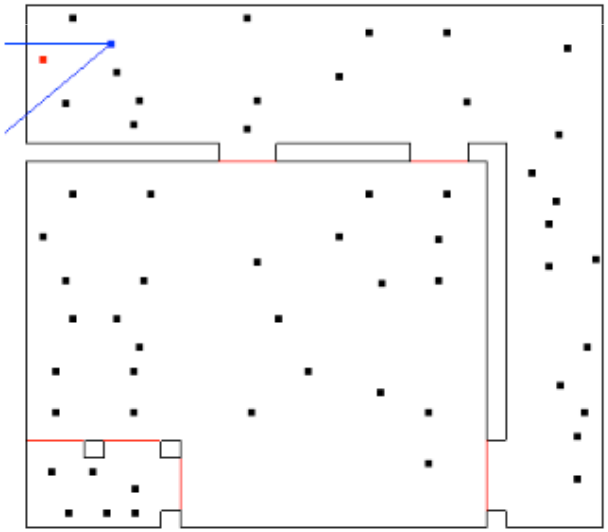
7 salas
10 portais

5 salas
8 portais

MAPA 3



MAPA 4



3 salas
6 portais

Resultados e discussões

Testes – Desempenho dos métodos

- Primeira bateria de testes

	Mapa 1	Mapa 2	Mapa 3	Mapa 4
Visão do observador	1,2447ms	1,4113ms	1,0848ms	1,5701ms
Mover observador	0,1437ms	0,0877ms	0,1850ms	0,1429ms

Resultados e discussões

Testes – Desempenho dos métodos

- Segunda bateria de testes

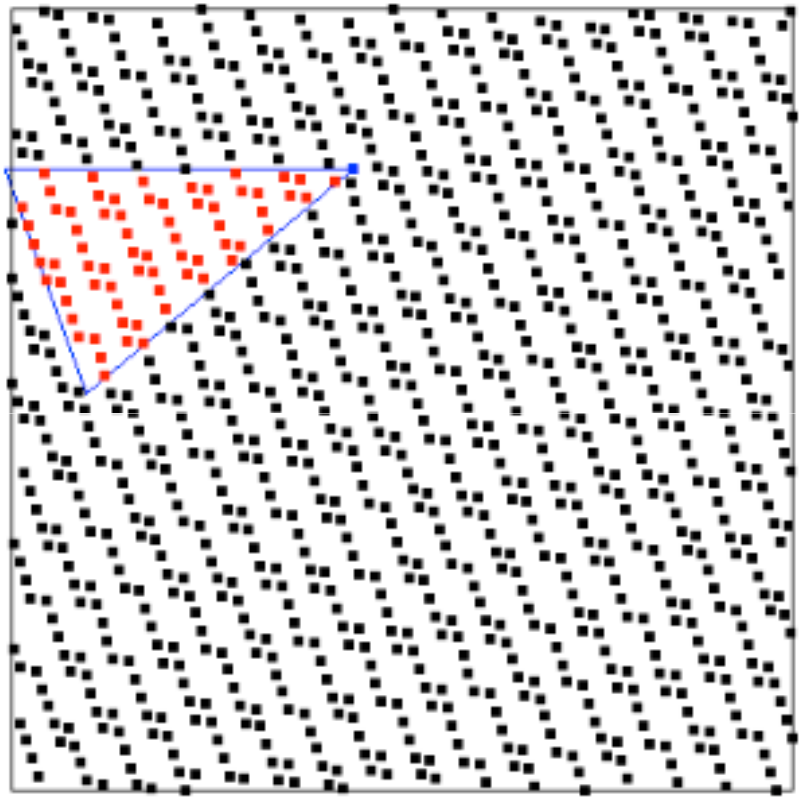
	Mapa 1	Mapa 2	Mapa 3	Mapa 4
Visão do observador	0,7155ms	0,8869ms	0,9440ms	1,0576ms
Mover observador	0,0767ms	0,1894ms	0,1803ms	0,1562ms

Resultados e discussões

Testes – Comparação entre os métodos

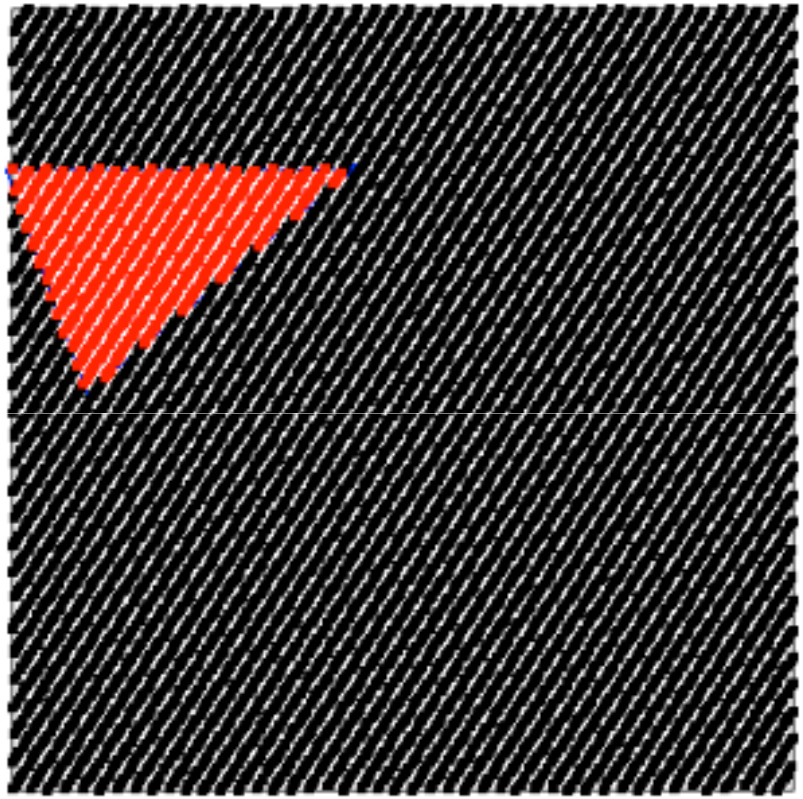
- Comparação dos algoritmos de teste de visibilidade

MAPA 5



1.000 pontos de interesse

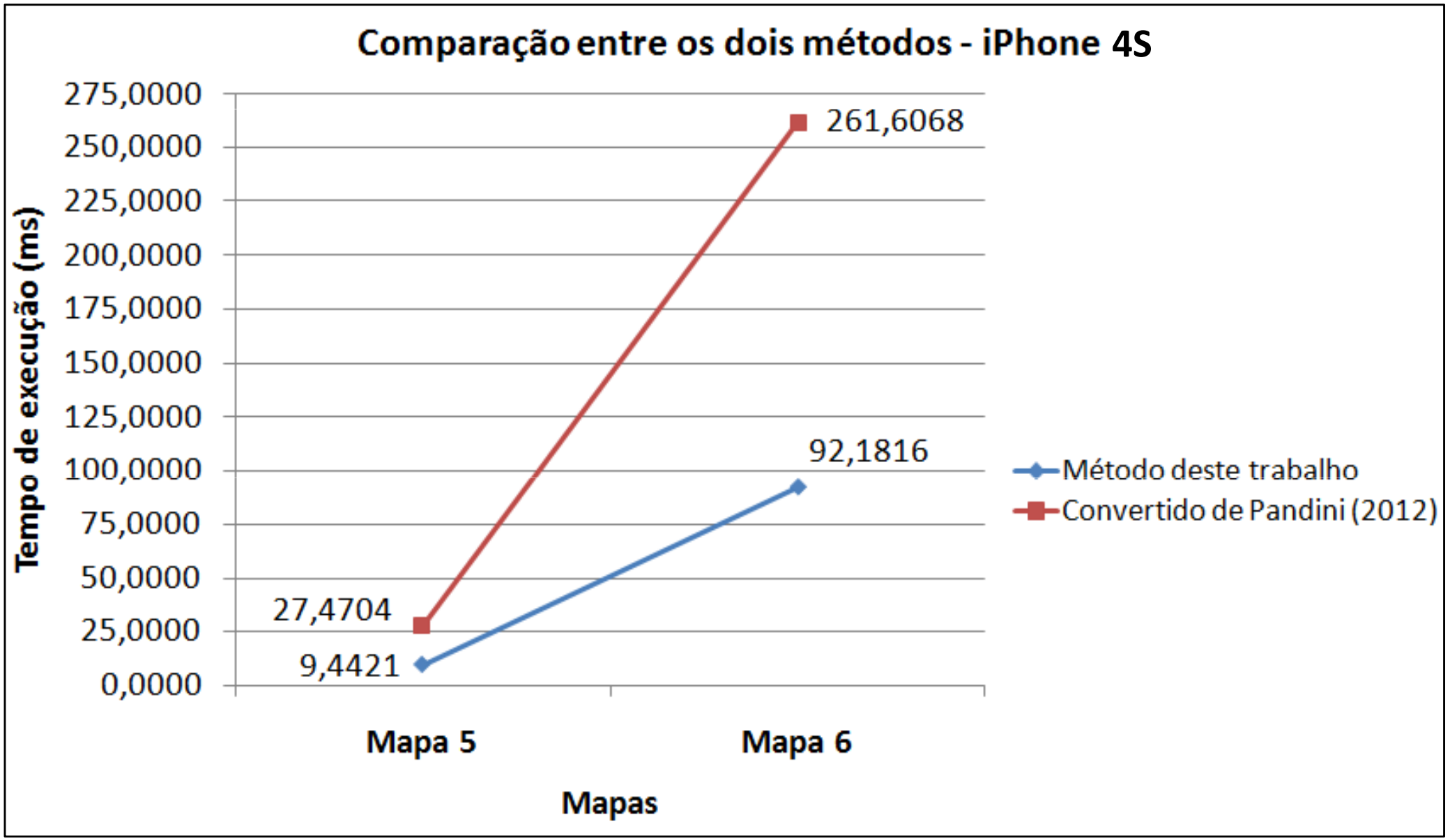
MAPA 6



10.000 pontos de interesse

Resultados e discussões

Testes – Comparação entre os métodos



Conclusão

- Melhor entendimento das técnicas
- Conhecimento em Objetive-C e OpenGL ES
- Técnicas utilizadas por Pandini (2012) serviram como base para buscar novas formas de resolver o problema
  - Montar estrutura da biblioteca deste trabalho
- Foi necessário desenvolver de outras maneiras as principais classes
- MJ3D: possibilidade de usar em Ambientes com Animação Comportamental na simulação da Percepção Visual
- Limitações
  - Portais que apontam para a mesma sala
- Resultados atingiram os objetivos propostos

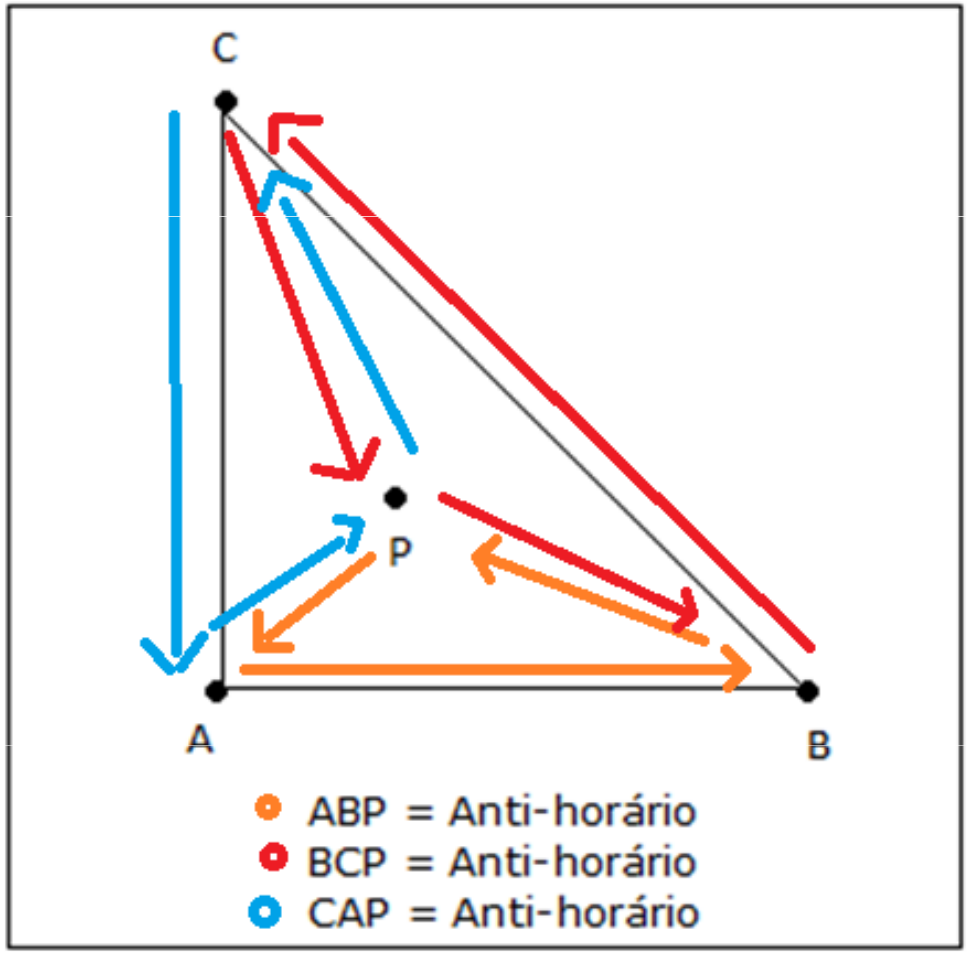
Extensões

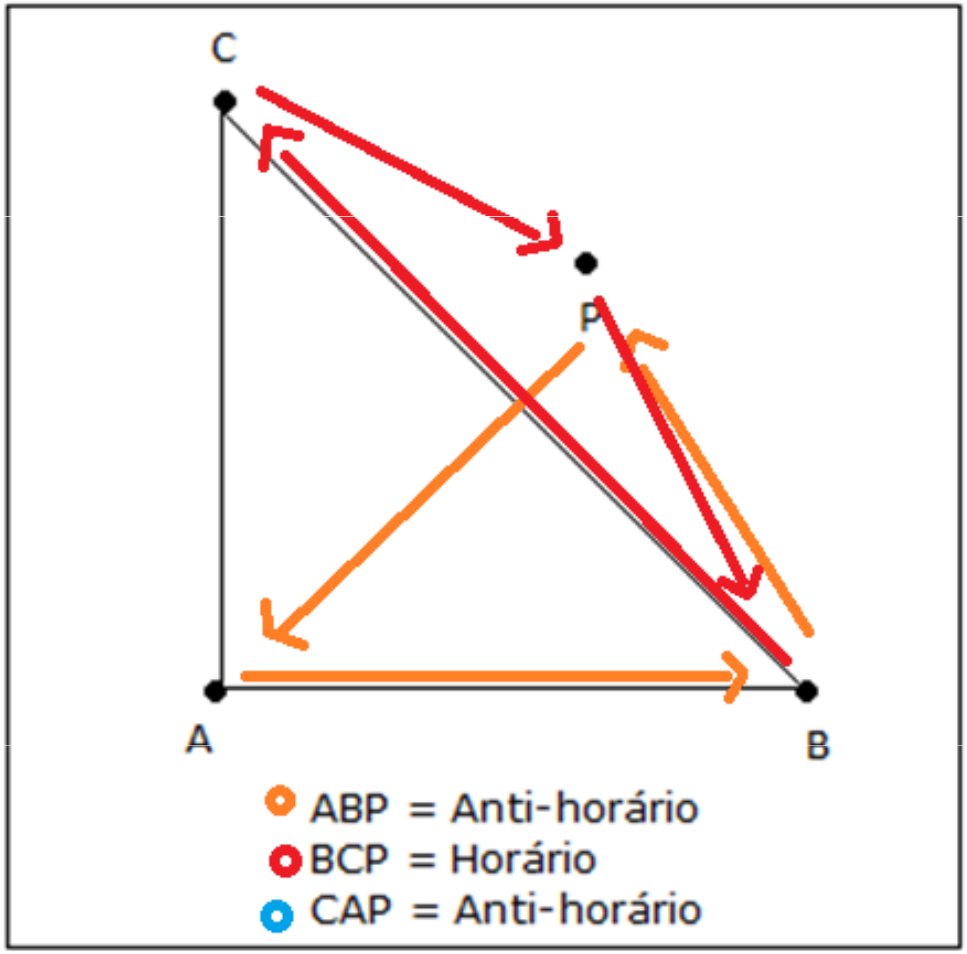
- Testes de visibilidade em 3D
- Mais de um observador no mapa
- Enxergar mais de um portal para a mesma sala
- Outras ferramentas de modelagem

DEMONSTRAÇÃO

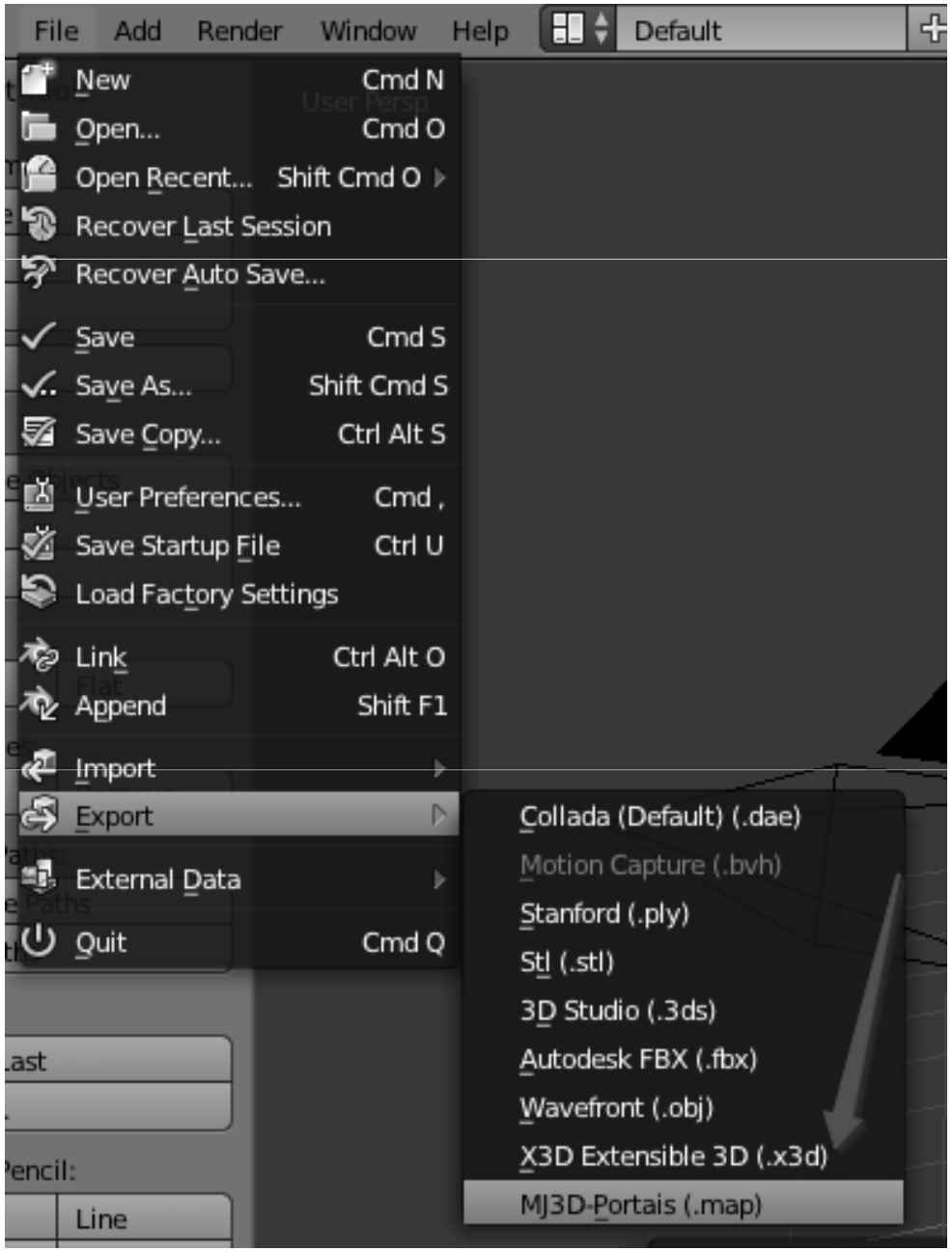
DEMONSTRAÇÃO

Ponto dentro do triângulo – Exemplo

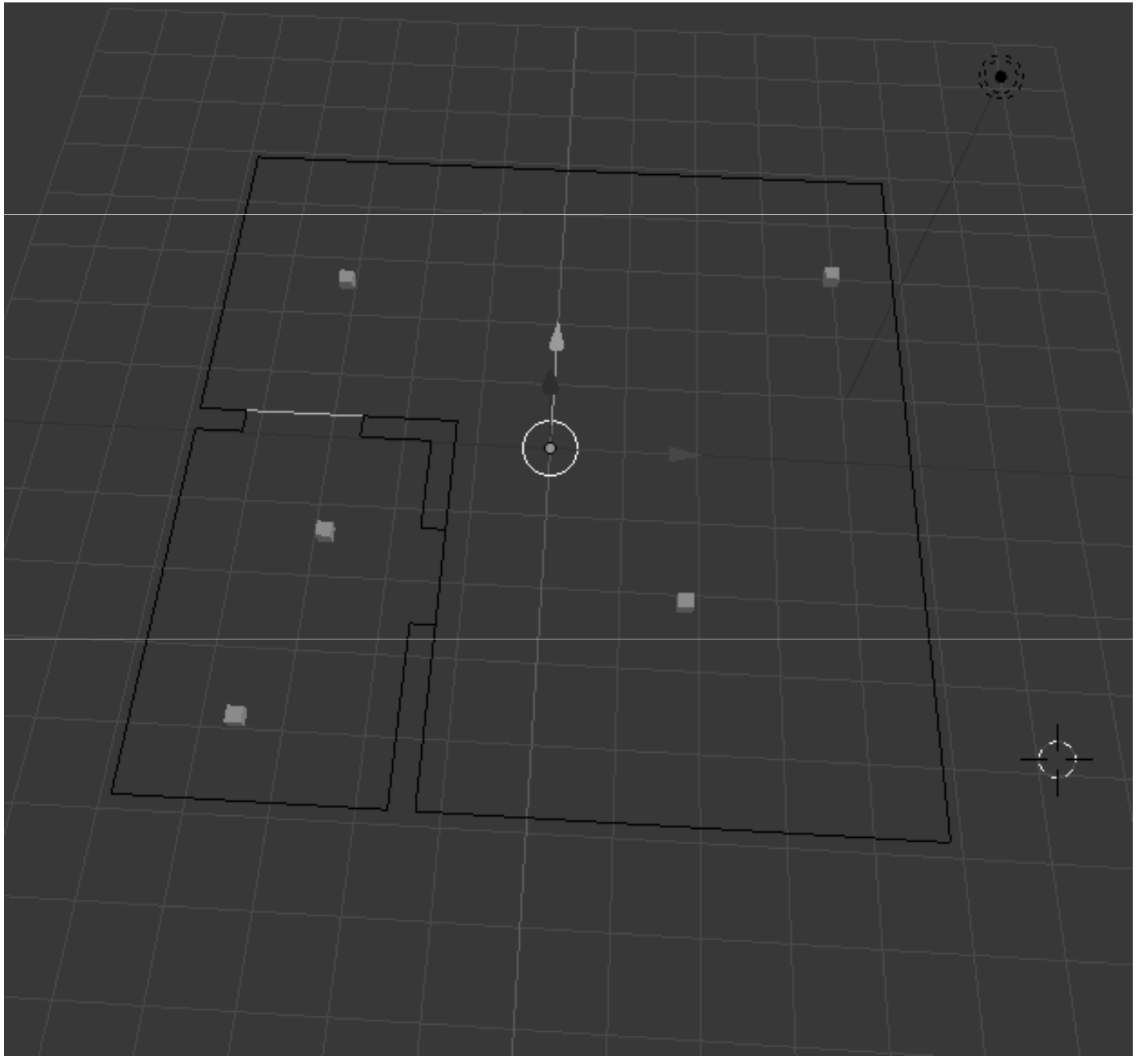




Add-on instalado no Blender



“Planta baixa” no Blender



Nome dos objetos no Blender