Reconstrutor de modelos 3D utilizando técnica de nível de detalhamento no iOS

FELIPE AUGUSTO IMIANOWSKY
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 discussão
Conclusão
Extensões

Introdução

- Dispositivos móveis
  - Presente no dia-a-dia das pessoas
  - iOS
- Nível de detalhamento ou LOD (*Level Of Detail*)
  - Técnica de *culling*
  - Jogos e simulações

Objetivos

Visualizar modelos 3D Wavefront OBJ

Aplicar LOD em modelos 3D em tempo real

Apresentar performance

Fundamentação teórica

Modelo Wavefront OBJ

OpenGL ES

Nível de detalhamento

Redução poligonal

Trabalhos correlatos

Modelo Wavefront OBJ

Modelo 3D

Simples e legível

Arquivos *.OBJ* e *.MTL*

Importação e exportação

Palavras chaves
v, *vn*, f, etc.

```
mtllib cube.mtl
usemtl red
v -1.000000 -1.000000 1.000000
v -1.000000 1.000000 1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
vn 0.577350 0.577350 -0.577350
vn 0.577350 -0.577350 -0.577350
vn -0.577350 -0.577350 -0.577350
vn -0.577350 0.577350 -0.577350
vn 0.577350 0.577350 0.577350
```

Presente na maioria das ferramentas 3D
Autodesk Maya, 3ds Max, Blender, Zbrush, etc.

OpenGL ES

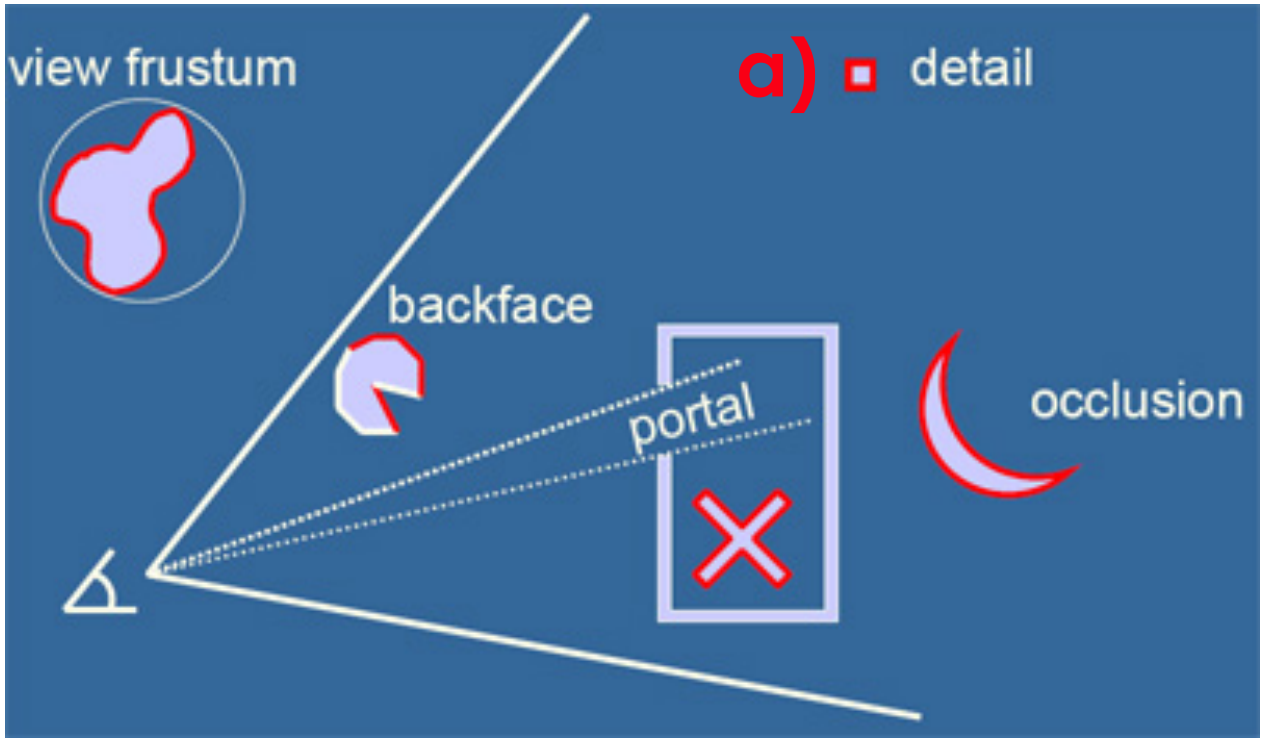
Conjunto de APIs para gráficos 3D

Dispositivos móveis e embarcados

OpenGL ES *Shading Language*

Nível de detalhamento

Técnicas de *culling*



Nível de detalhamento

Modelo menos detalhado

Algoritmos executam em 3 partes
- Geração, seleção e escolha

3 tipos de LOD
- Discreto, contínuo e dependente de visão

Redução poligonal

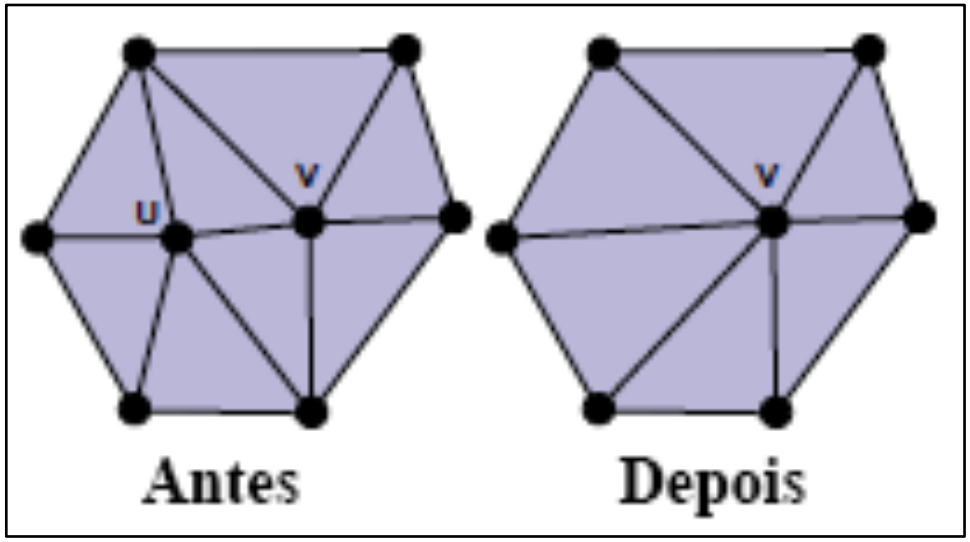
Stan Melax (1998)

Colapso de aresta (Hoppe 1996)

Heurística de seleção de aresta simples

Colapso de aresta

Move *u* para *v*



Heurística

Tamanho da aresta

Termo de curvatura

Prioriza áreas planares com bastante vértices

$$\mathbf{cost}(u,v) = \|u - v\| \times \max_{f \in Tu} \left\{ \min_{n \in Tuv} \left\{ \left(1 - f.normal \bullet n.normal\right) \div 2 \right\} \right\}$$

Melax (1998)

Trabalhos correlatos

Reconstrutor de modelos 3D utilizando técnicas de nível de detalhamento (Piske, 2008)

AI Aardvark (simulador de aviões)

Unity 3D

Reconstrutor de LOD de Piske

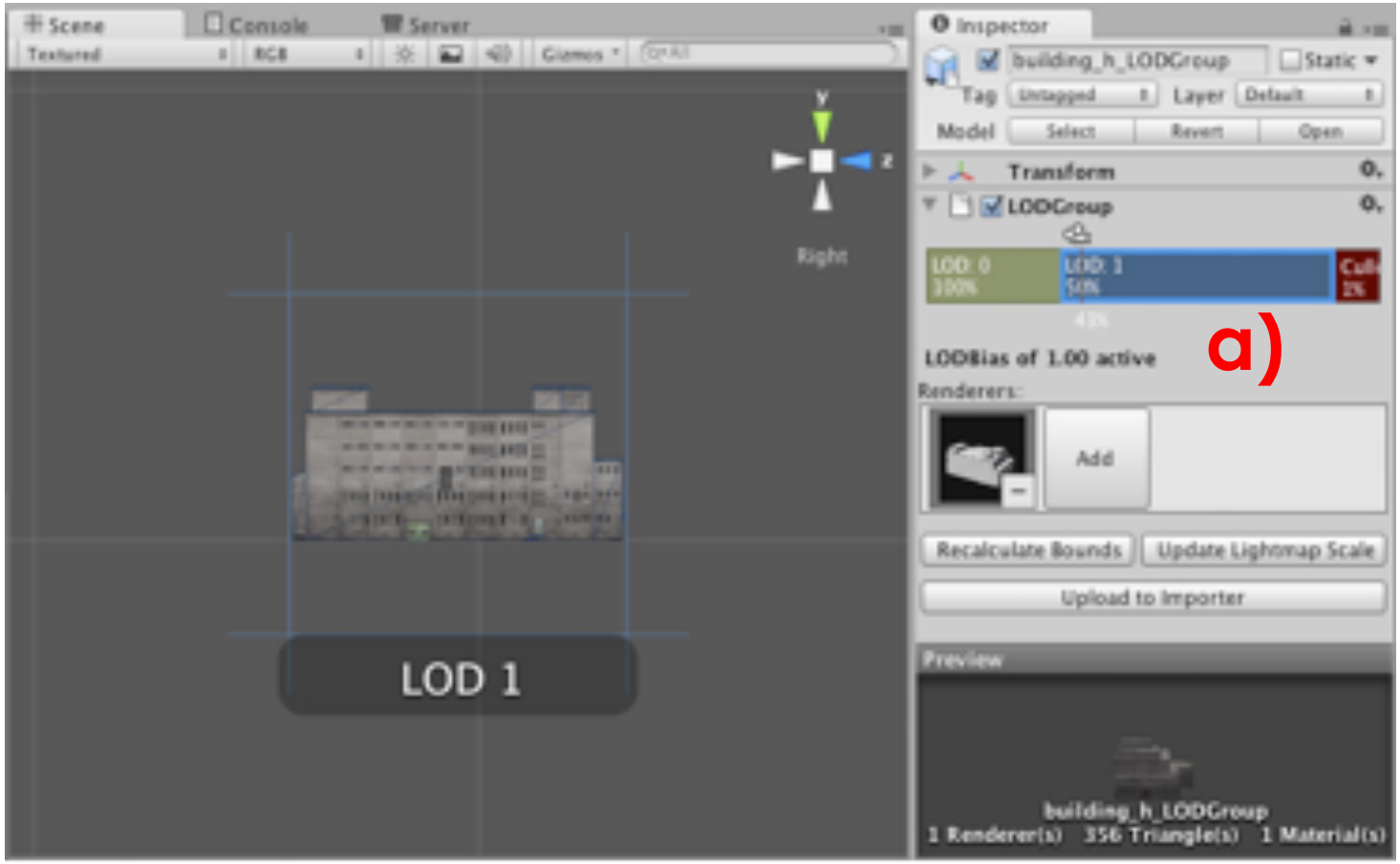
Modelos MD2 (Quake 2)
Mobile 3D Game Engine (M3GE)
Algoritmo de Redução Poligonal
LOD pré-processado

AI Aardvark

Simulador de vôos de aviões

Unity 3D

Motor de jogos e editor de cenas
LOD discreto



Desenvolvimento

Casos de uso

Diagramas

Implementação

Casos de uso

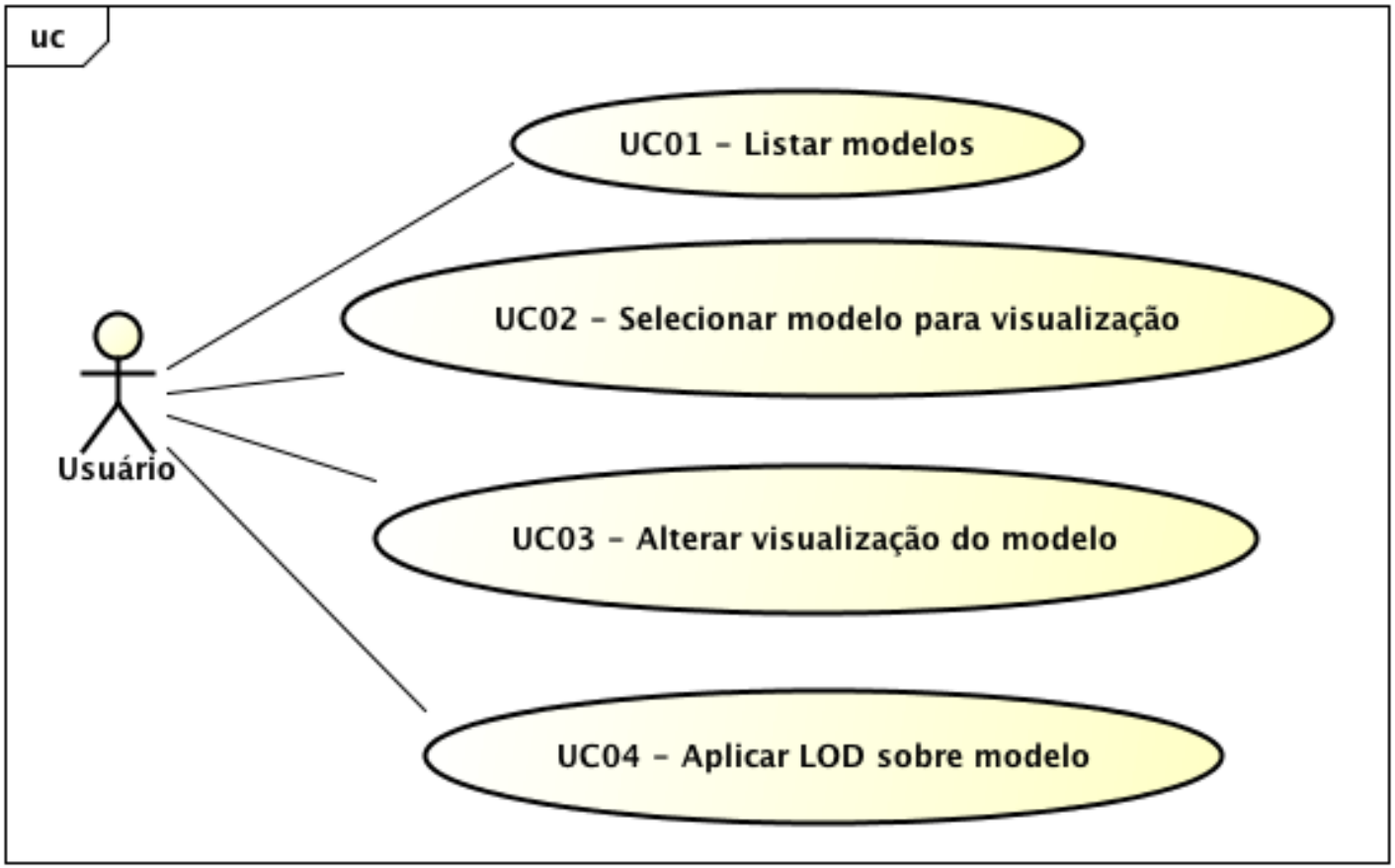

uc
UC01 – Listar modelos
UC02 – Selecionar modelo para visualização
Usuário
UC03 – Alterar visualização do modelo
UC04 – Aplicar LOD sobre modelo

Diagrama: Pacotes

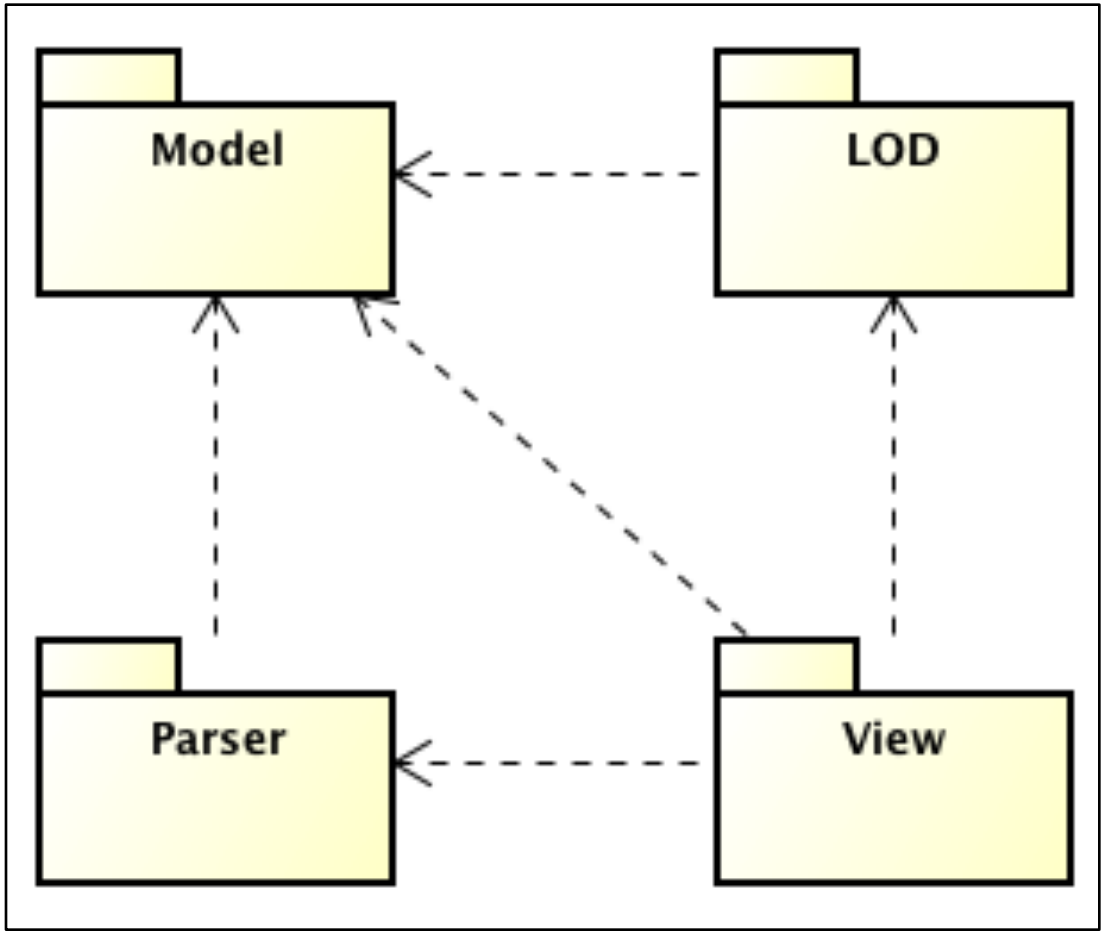


Diagrama classe: LOD

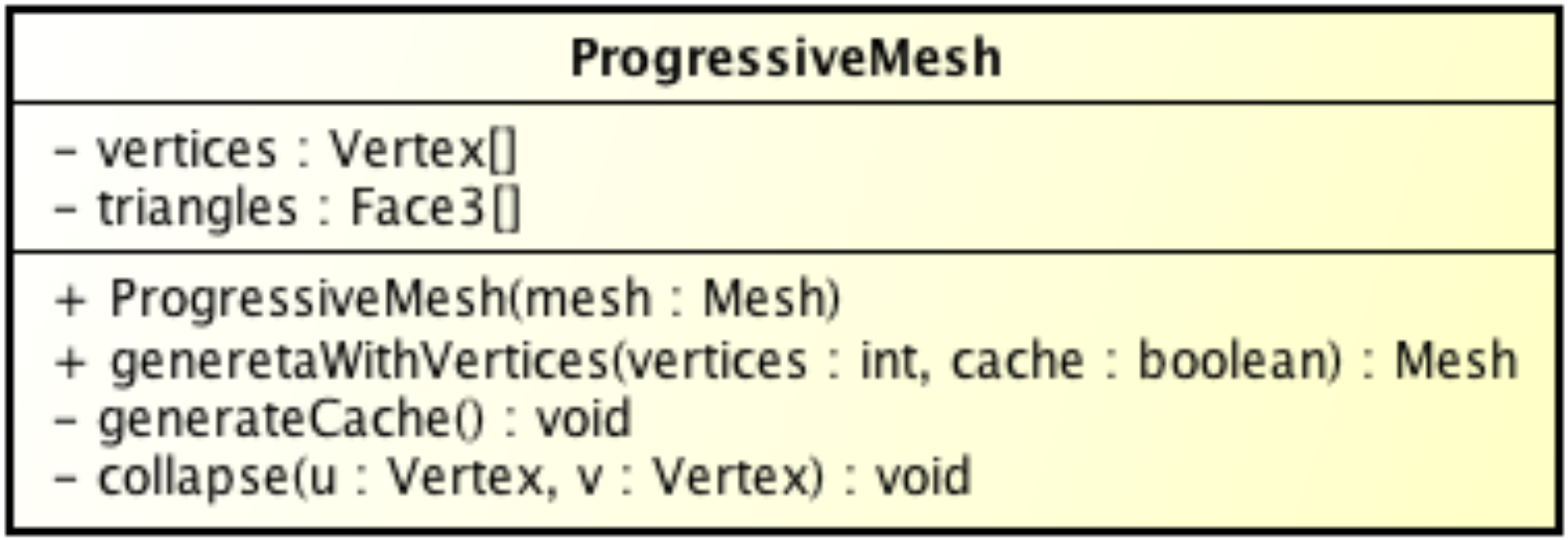


Diagrama classe: Model

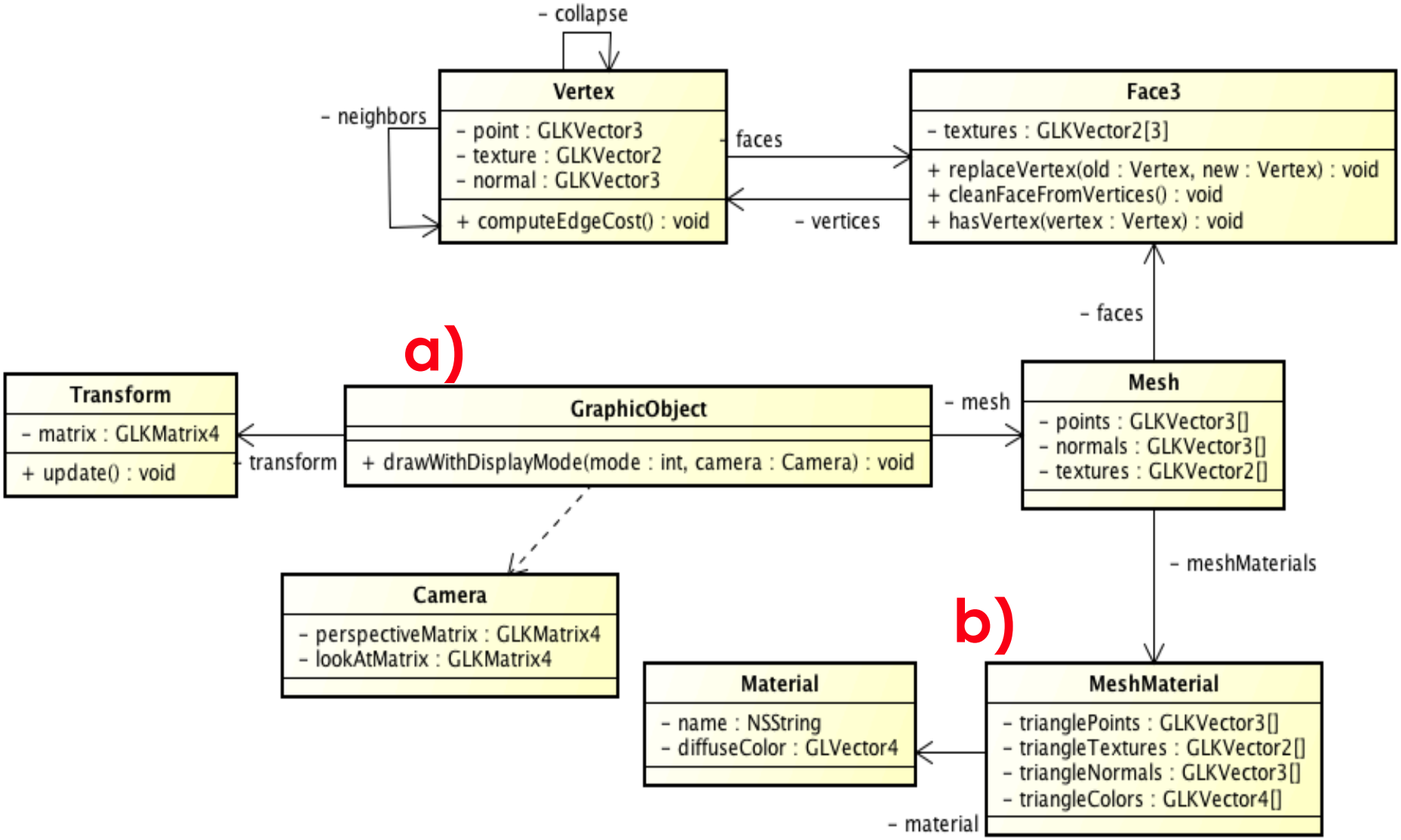


Diagrama classe: Parser

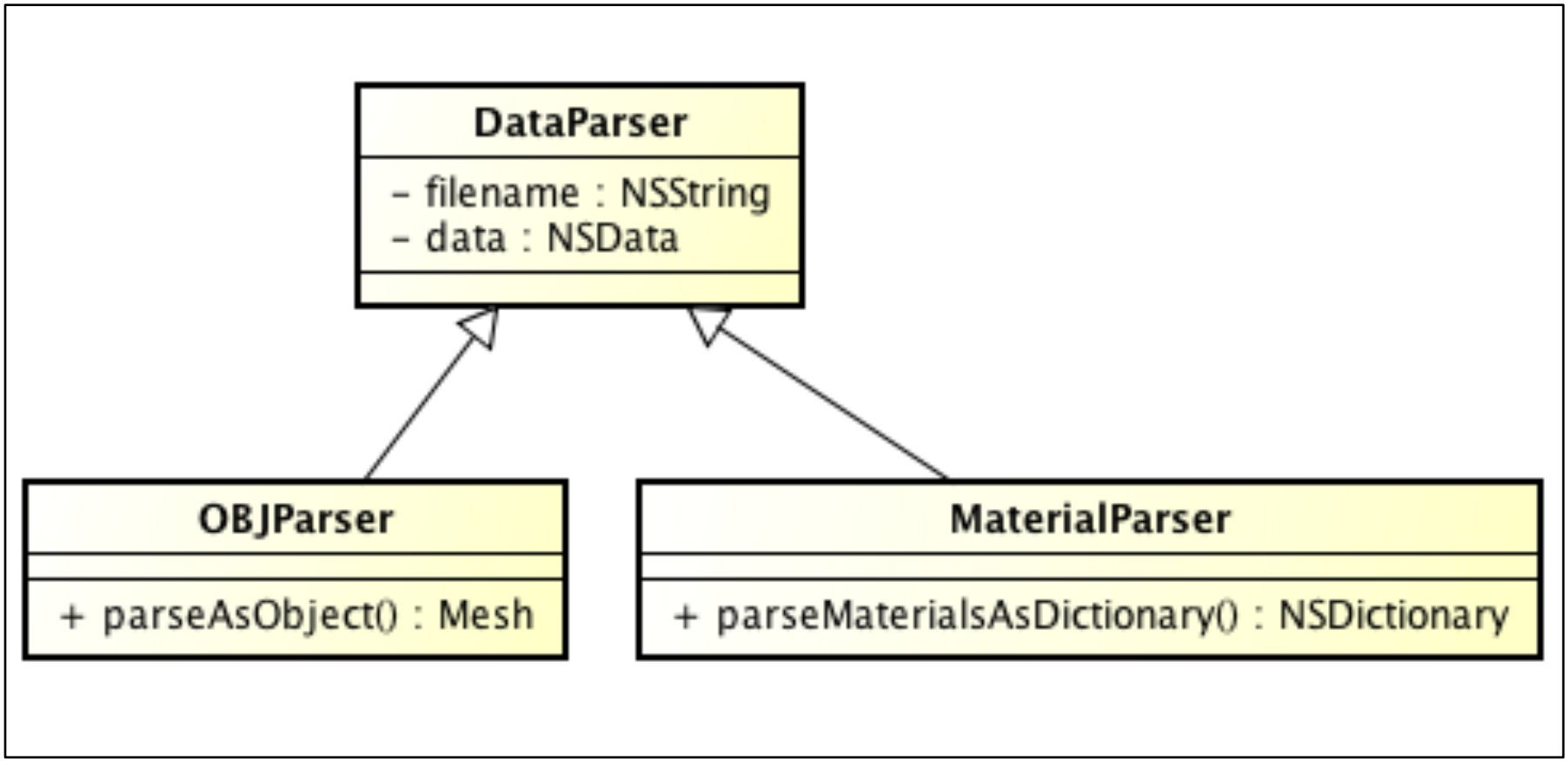


Diagrama classe: View

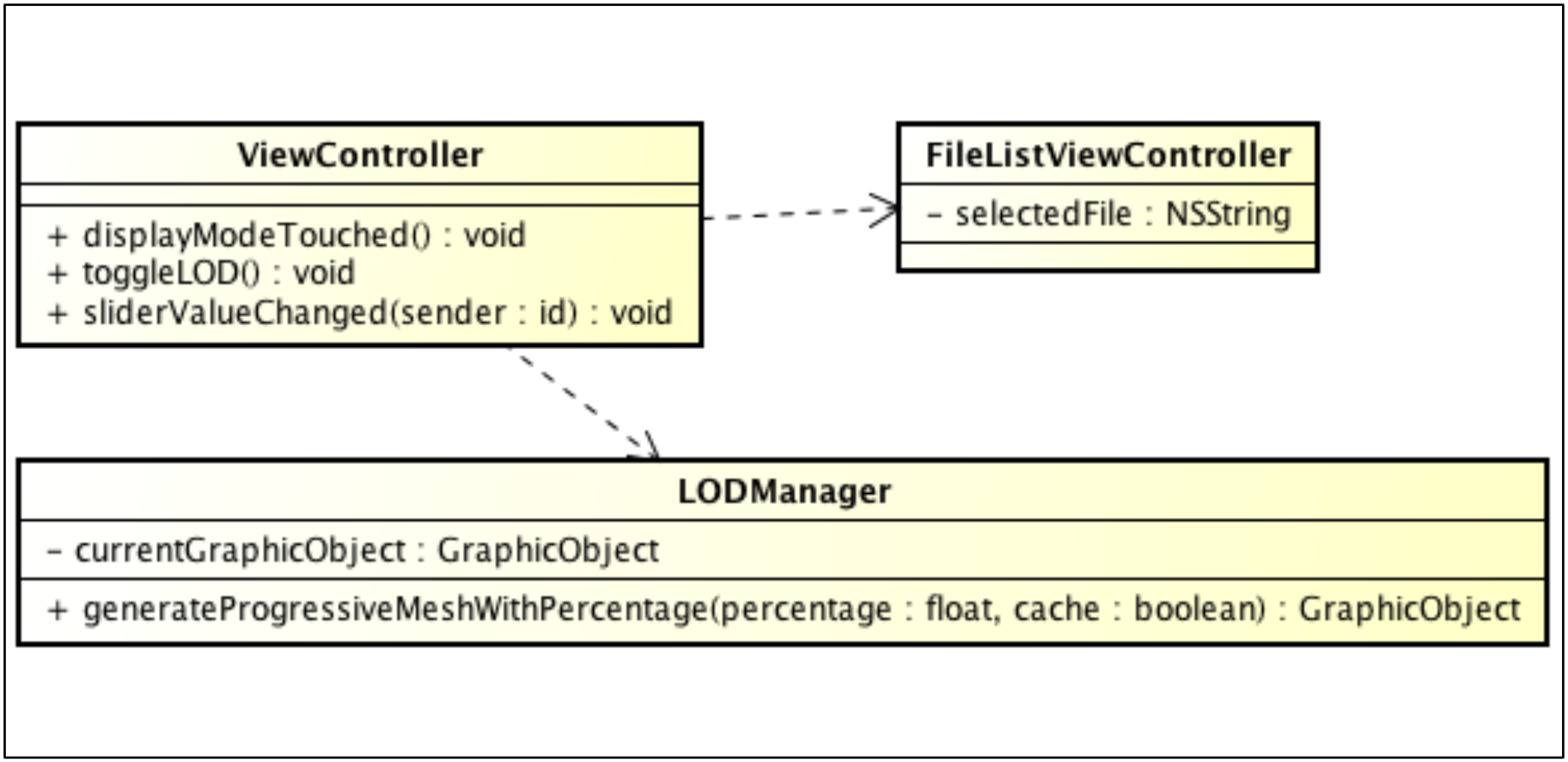
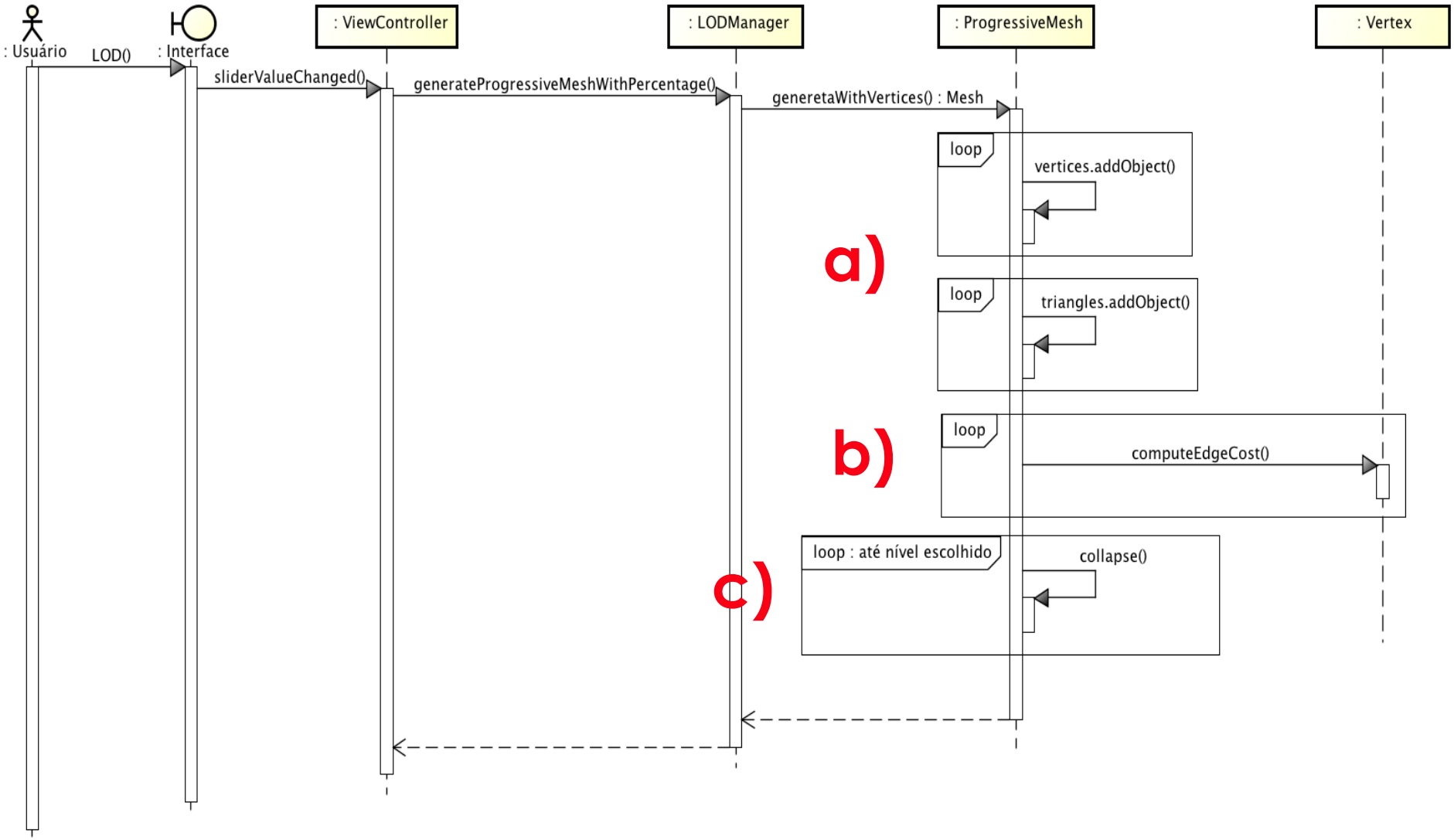


Diagrama sequência: LOD



Implementação

Orientação a objetos

Linguagem Objective-C

OpenGL ES 2.0

Xcode

Implementação: LOD

a)

b)

c)

```
- (Mesh *)generateMeshWithoutCacheWithVertices:(NSUInteger)vertices
{
    Mesh *originalMesh = self.originalMesh;

    self.vertices = [NSMutableArray arrayWithCapacity:originalMesh.points.count];

    // adiciona os pontos da malha original numa nova lista
    for (NSUInteger i = 0; i < originalMesh.points.count; i++) {...}

    self.triangles = [NSMutableArray arrayWithCapacity:originalMesh.faces.count];

    // adiciona os triangulos da malha original numa nova lista
    for (Face3 *face in originalMesh.faces) {...}

    // calcula o custo de colapso dos vertices e seus candidatos
    for (Vertex *vertex in self.vertices) {
        [vertex computeEdgeCost];
    }

    while (self.vertices.count > vertices) {
        // recupera o vertice com menor custo de colapso
        Vertex *mn = [self minimumCostEdge];
        [self collapse:mn v:mn.collapse];
    }

    Mesh *newMesh = [[Mesh alloc] init];

    for (Face3 *face in self.triangles) {
        [newMesh addFace:face];
    }

    return newMesh;
}
```


Cancel
AntLion.obj
Barrel Explosive.obj
buggy.obj
classic-lamppost.obj
Combine_dropship.obj
dog.obj
Rollermine.obj


FPS: 60.0
Vertices: 39
Faces: 74
Cache
OFF
40%
LOD

Resultados e discussão

Trabalhos / Características	iOBJ	Reconstrutor de LOD de Piske	Simulador de aviões	Unity 3D
Importação modelos 3D	**Sim**	Sim	Sim	Sim
LOD Discreto	**Não**	Não	Sim	Sim
LOD Contínuo	**Sim**	Sim	?	Não
Confgurar qtde de vértices LOD	**Sim**	Sim	Não	Não
Suporte dispositivo móvel	**Sim**	Sim	Não	Sim
LOD calculado em tempo real	**Sim**	Não	Não	Não

Resultados e discussão

Aparelhos testados

iPhone 3GS
256MB de memória eDRAM
processador ARM Cortex-A8 de 600 MHz
processador gráfico PowerVR

iPad 4
1 GB de memória DDR2 RAM
processador *dual core* Apple Swift de 1.4 GHz
processador gráfico Quad-core PowerVR

Resultados e discussão

Número vértices	Memória antes		Memória após LOD 50%		Memória após LOD 100%	
-	Normal	Cache	Normal	Cache	Normal	Cache
98	1.43MB	1.43MB	1.62MB	1.88MB	1.72MB	2.02MB
1232	3.41MB	3.42MB	4.30MB	8.94MB	5.60MB	10.06MB
4210	9.06MB	9.06MB	12.09MB	29.48MB	16.71MB	33.14MB

Resultados e discussão

Número vértices	Memória antes		Memória após LOD 50%		Memória após LOD 100%	
-	Normal	Cache	Normal	Cache	Normal	Cache
98	1.43MB	1.43MB	1.62MB	1.88MB	1.72MB	2.02MB
1232	3.41MB	3.42MB	4.30MB	8.94MB	5.60MB	10.06MB
4210	9.06MB	9.06MB	**12.09MB**	**29.48MB**	16.71MB	33.14MB

Resultados e discussão

Obs.: Valores em *ms* no iPad 4

Tipos / Vert.	Normal		Cache	
	1º LOD	2º LOD	1º LOD	2º LOD
98	432	148	718	69
1232	3.463	1.306	6.928	901
4210	16.546	4.515	33.441	3.285

Resultados e discussão

Obs.: Valores em *ms* no iPad 4

Tipos / Vert.	Normal		Cache	
	1º LOD	2º LOD	1º LOD	2º LOD
98	432	148	718	69
1232	3.463	1.306	6.928	901
4210	**16.546**	4.515	**33.441**	3.285

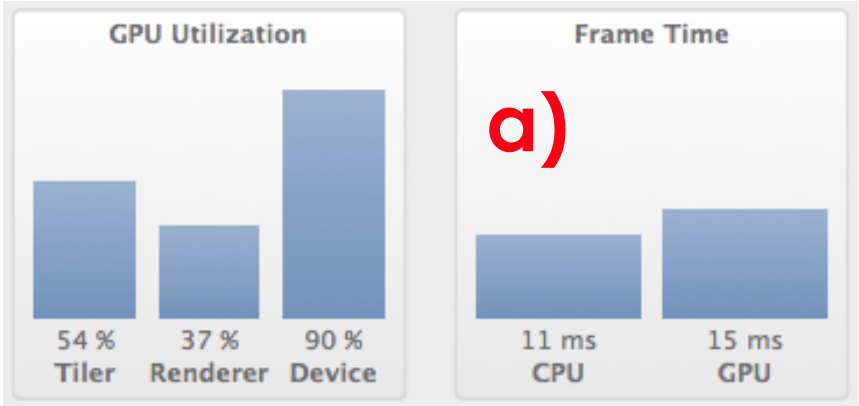
Resultados e discussão

Obs.: Valores em *ms* no iPad 4

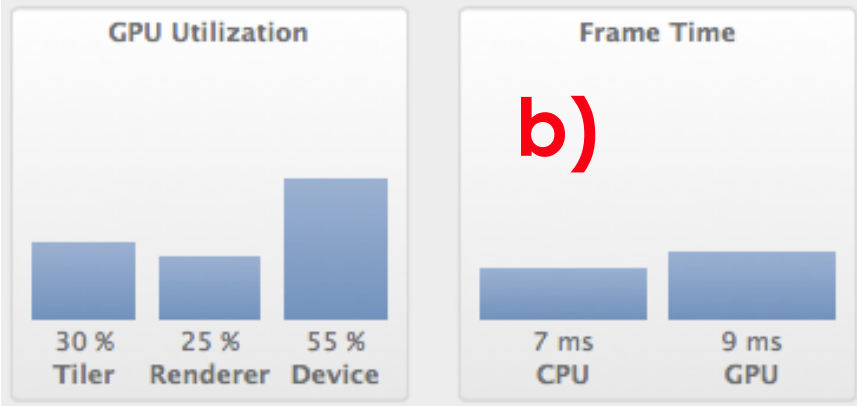
Tipos / Vert.	Normal		Cache	
	1º LOD	2º LOD	1º LOD	2º LOD
98	432	148	718	69
1232	3.463	1.306	6.928	901
4210	16.546	**4.515**	33.441	**3.285**

Resultados e discussão

SEM LOD



COM LOD



Resultados e discussão

Modelos desconexos

Conclusão

Biblioteca atende os requisitos propostos

Boas reconstruções

Bom uso dos recursos do dispositivo

Destaca-se por aplicar LOD em tempo real

Cassaniga (2013) e Oliveira (2013) utilizaram o leitor *OBJ*

Problemas:

- Dispositivos com recurso limitado
- Modelos desconexos

Extensões

Fórmula do custo de colapso de aresta

Aprimoramento e estudo do algoritmo de cache

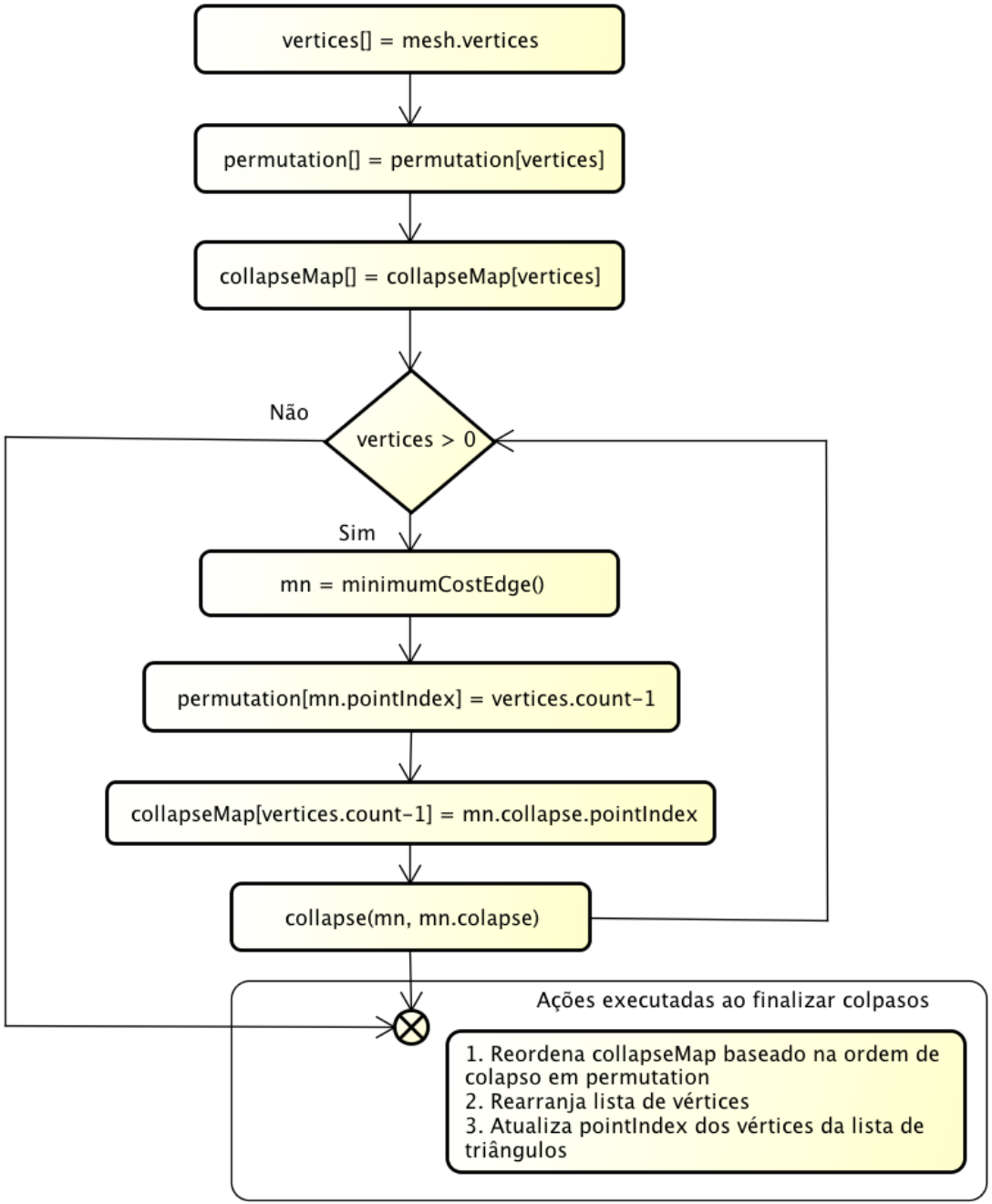
Carregamento de modelos em outros formatos

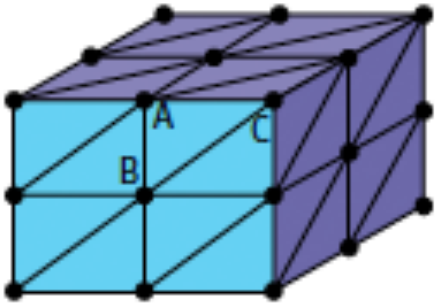
Aplicação de outros algoritmos de LOD
- Comparação
- Modelos desconexos

Aplicação em outras plataformas (Android, Windows Phone)

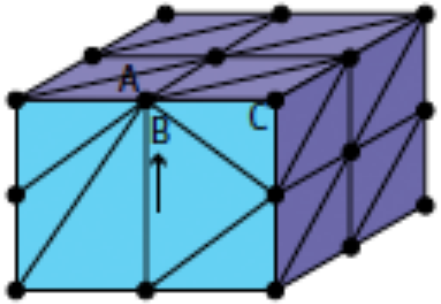
LOD dependente de visão

Demonstração

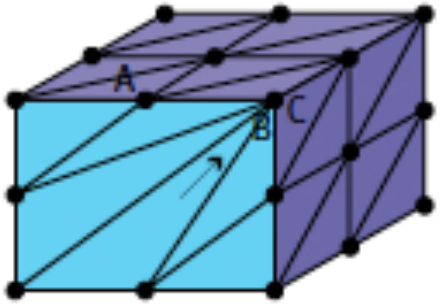

vertices[] = mesh.vertices
permutation[] = permutation[vertices]
collapseMap[] = collapseMap[vertices]
Não
vertices > 0
Sim
mn = minimumCostEdge()
permutation[mn.pointIndex] = vertices.count-1
collapseMap[vertices.count-1] = mn.collapse.pointIndex
collapse(mn, mn.colapse)
Ações executadas ao finalizar colpasos
1. Reordena collapseMap baseado na ordem de colapso em permutation
2. Rearranja lista de vértices
3. Atualiza pointIndex dos vértices da lista de triângulos



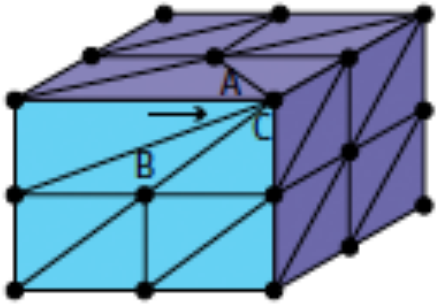
Original



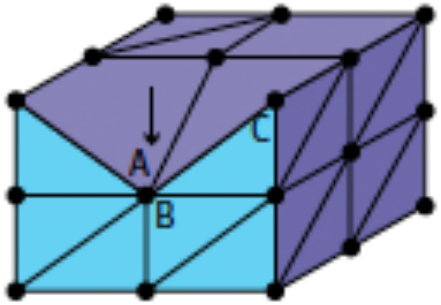
B para A



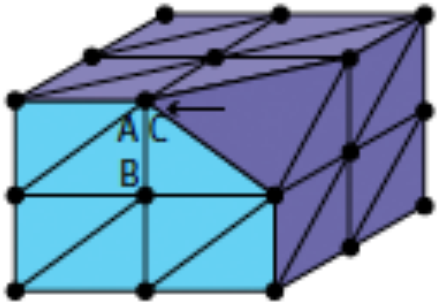
B para C



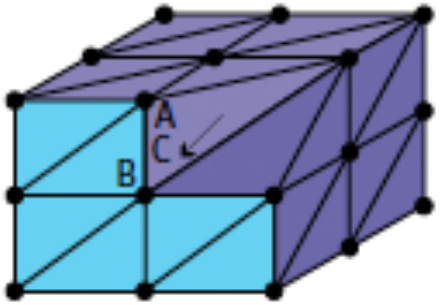
A para C



A para B



C para A



C para B