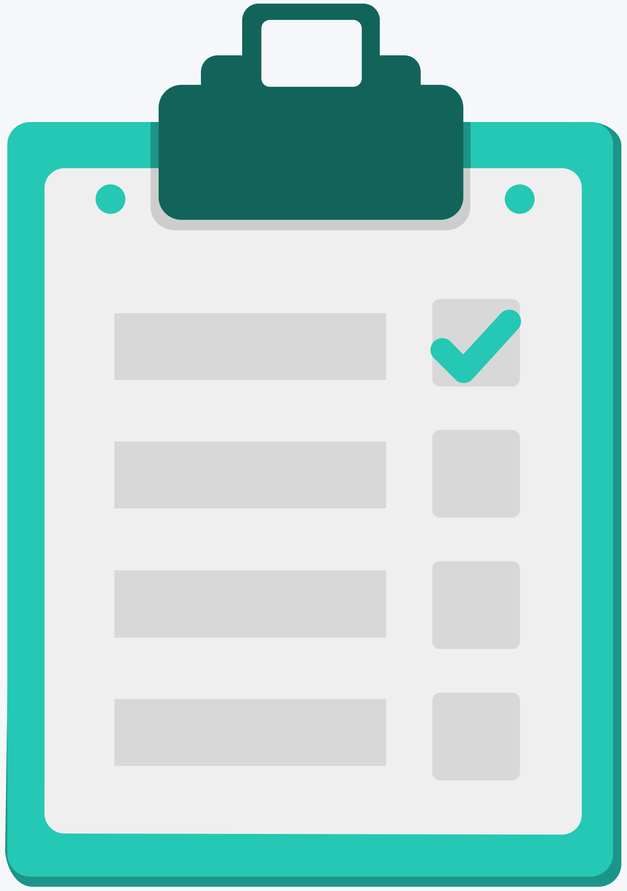
QtCon BR

Computação Gráfica com Qt 3D

Sandro S. Andrade
sandroandrade@kde.org

IFBA/KDE

Objetivos



1. Apresentar os principais conceitos, fundamentos e aplicações da Computação Gráfica.
2. Apresentar os principais recursos do Qt3D para renderização de modelos 3D e definição de câmeras, materiais, luzes e texturas.
3. Apresentar os principais recursos do Qt3D para interação com cenas 3D e programação de animações.
4. Proporcionar vivências práticas sobre os tópicos acima.

whoareyou?

QtCon BR

whoami?

Professor no Instituto Federal de Educação, Ciência e Tecnologia da Bahia (IFBA)
Colaborador nas comunidades Qt e KDE há 10 anos
Desenvolvedor/Arquiteto C++ e Qt há 18 anos

sandroandrade@kde.org

sandroandrade.org

@andradesandro

Agenda

01

FUNDAMENTOS DE COMPUTAÇÃO GRÁFICA

Computação Gráfica, Processamento de Imagem, Visão Computacional, pipeline gráfico e modelo de câmera.

02

INTRODUÇÃO AO Qt3D

Objetivos, módulos principais, conceitos importantes, arquitetura, Entity-Component-Systems (ECS), APIs C++ e QML.

03

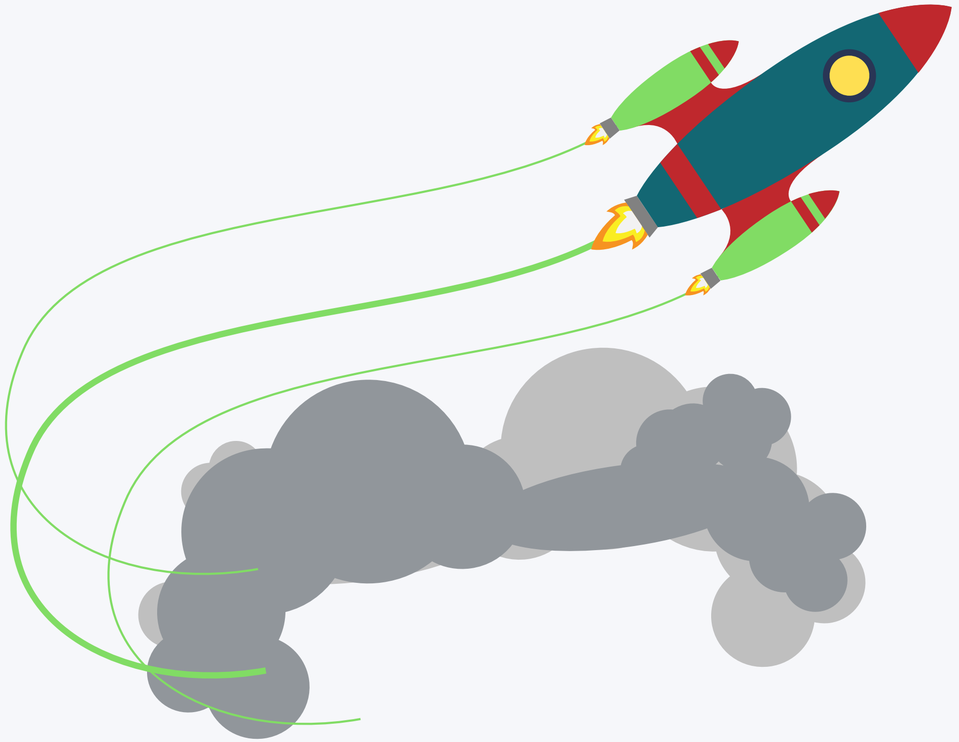
GEOMETRIA E TRANSFORMAÇÕES

Malhas poligonais primitivas, importando malhas, transformações..

04

RENDERIZAÇÃO E ANIMAÇÕES

Materiais, luzes, PBR, animações..



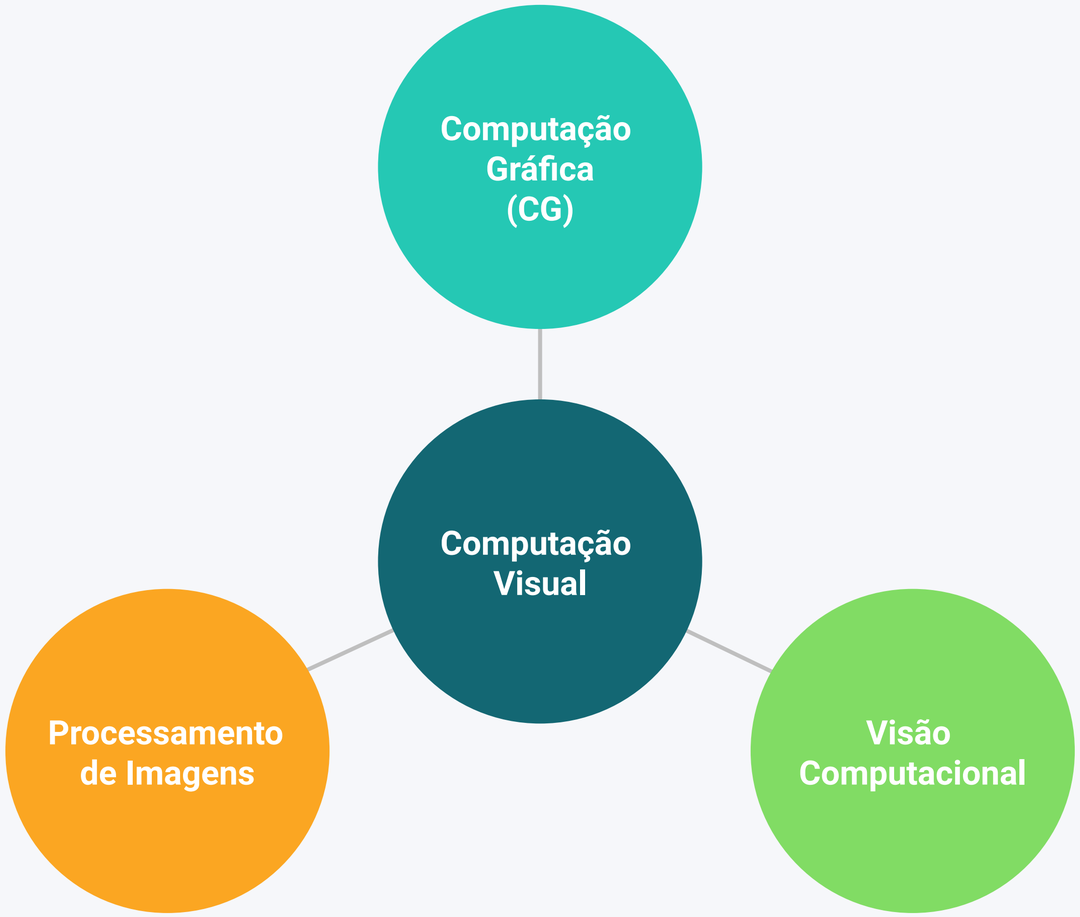
QtCon BR

FUNDAMENTOS DE COMPUTAÇÃO GRÁFICA

Computação Gráfica, Processamento de Imagem, Visão Computacional, pipeline gráfico e modelo de câmera

QtCon BR

Computação Visual



Computação Visual

É a área da Ciência da Computação que estuda como desenvolver sistemas computacionais onde imagens exercem papel fundamental.

COMPUTAÇÃO GRÁFICA

Dados → Imagem

PROCESSAMENTO DE IMAGENS

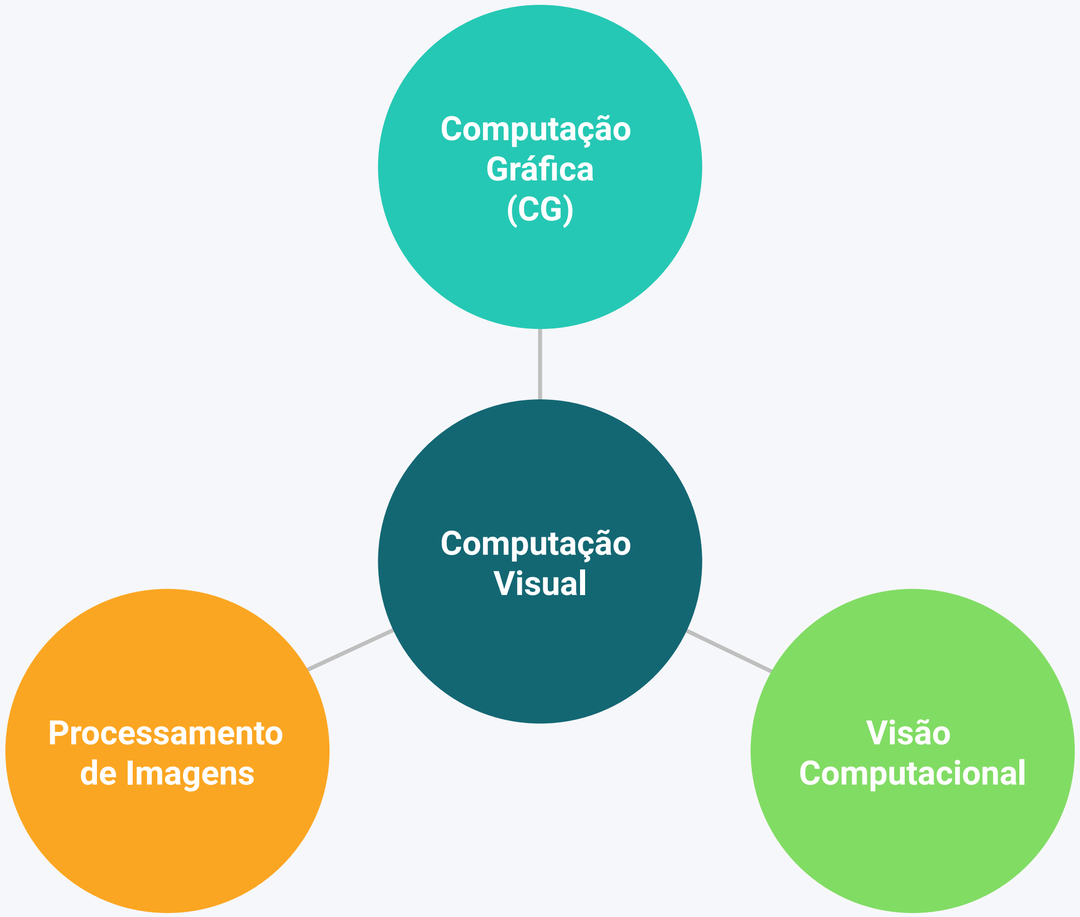
Imagem → Imagem

VISÃO COMPUTACIONAL

Imagem → Dados

QtCon BR

Computação Visual



Computação Visual

É a área da Ciência da Computação que estuda como desenvolver sistemas computacionais onde imagens exercem papel fundamental.

COMPUTAÇÃO GRÁFICA

Dados → Imagem

PROCESSAMENTO DE IMAGENS

Imagem → Imagem

VISÃO COMPUTACIONAL

Imagem → Dados

QtCon BR

Aplicações da Computação Gráfica

01

JOGOS E CINEMA

2D e 3D. Renderização e interação em tempo-real no caso de jogos. Renderização off-line no caso de cinema.

02

REALIDADE VIRTUAL E AUMENTADA

Diversas aplicações: jogos, terapias, educação.

03

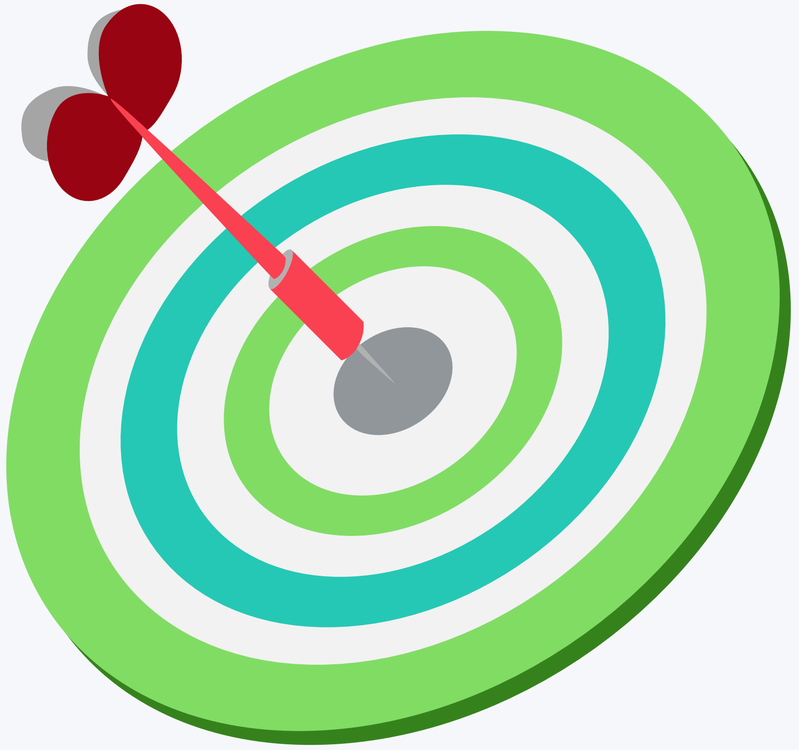
MEDICINA, CAD E ENGENHARIAS

Exames baseados em imagem, simulação de cirurgias. Projetos arquiteturais. Inspeção de turbinas de aviões.

04

HCI

Indústria automobilística, jogos, totens e outros displays.



QtCon BR

Renderização

Renderização

Processo automático de geração de uma imagem (fotorealista ou não) a partir de um modelo 2D ou 3D.

Diversas técnicas disponíveis:

Forward Render

Deferred Render

Ray Tracing

Visualização Volumétrica

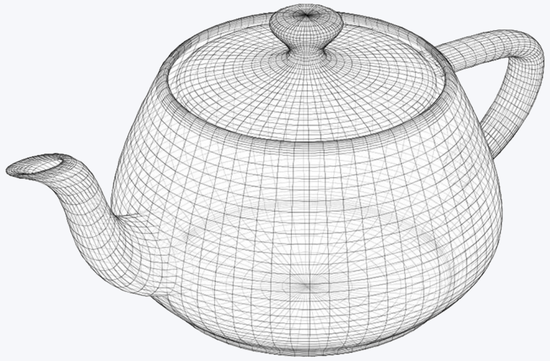
Radiosidade

Marching Cubes

Pipeline Gráfico

Dados de entrada = malhas poligonais 3D

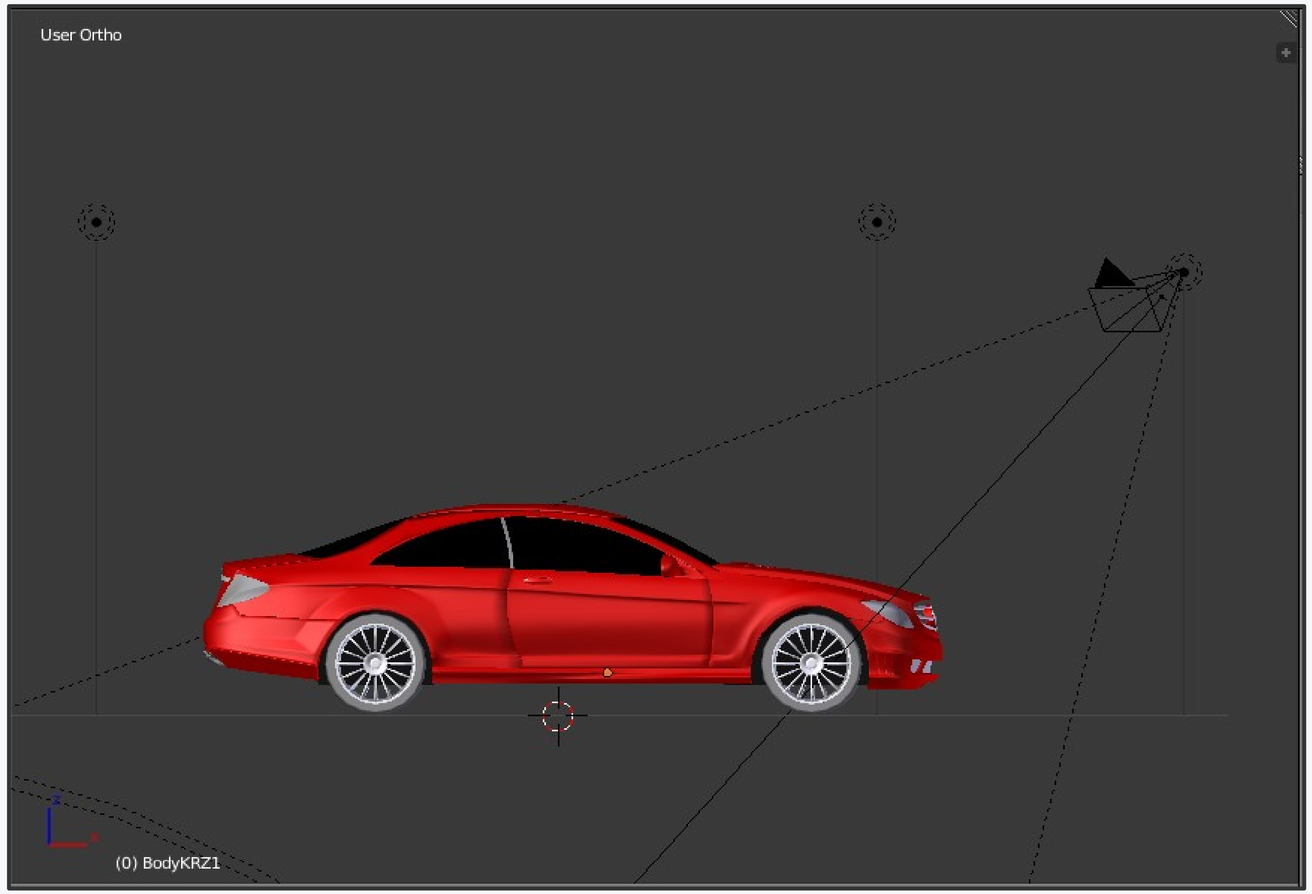
Geralmente formadas por triângulos.



1. Transformação de modelagem
2. Iluminação
3. Transformação de Câmera
4. Transformação de Projeção
5. Clipping
6. Transformação para Viewport
7. Rasterização

Transformação de Modelagem

Posiciona os diversos modelos 3D na cena (mundo).



Iluminação

Realiza os cálculos das funções de iluminação, considerando todas as fontes de luz presentes na cena.

Diversos tipos de luz podem estar presentes:

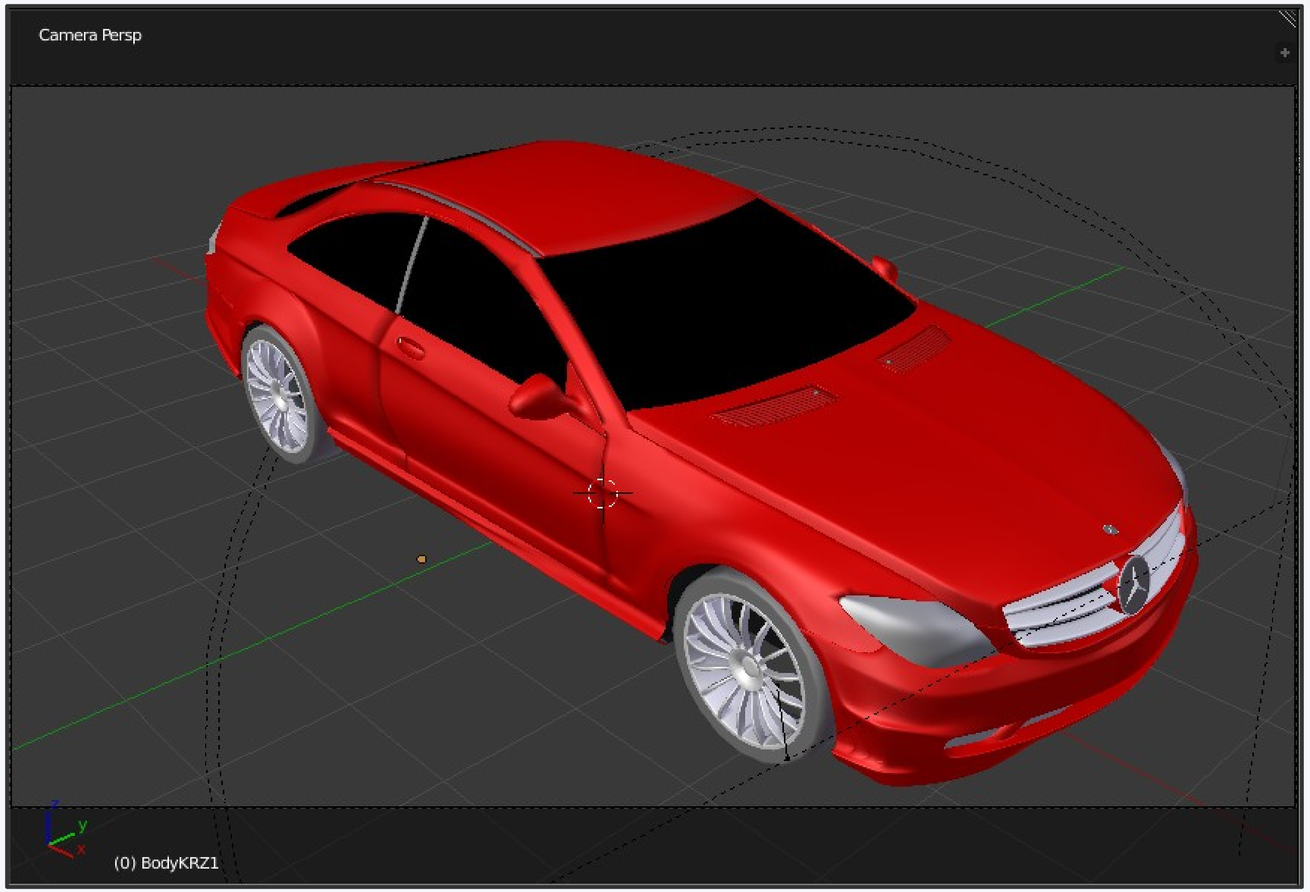
- Luz pontual
- Luz direcional
- Spot

Diversos modelos de iluminação podem ser utilizados:

- Lambert
- Phong
- Radiosidade
- HDRI

Transformação de Câmera

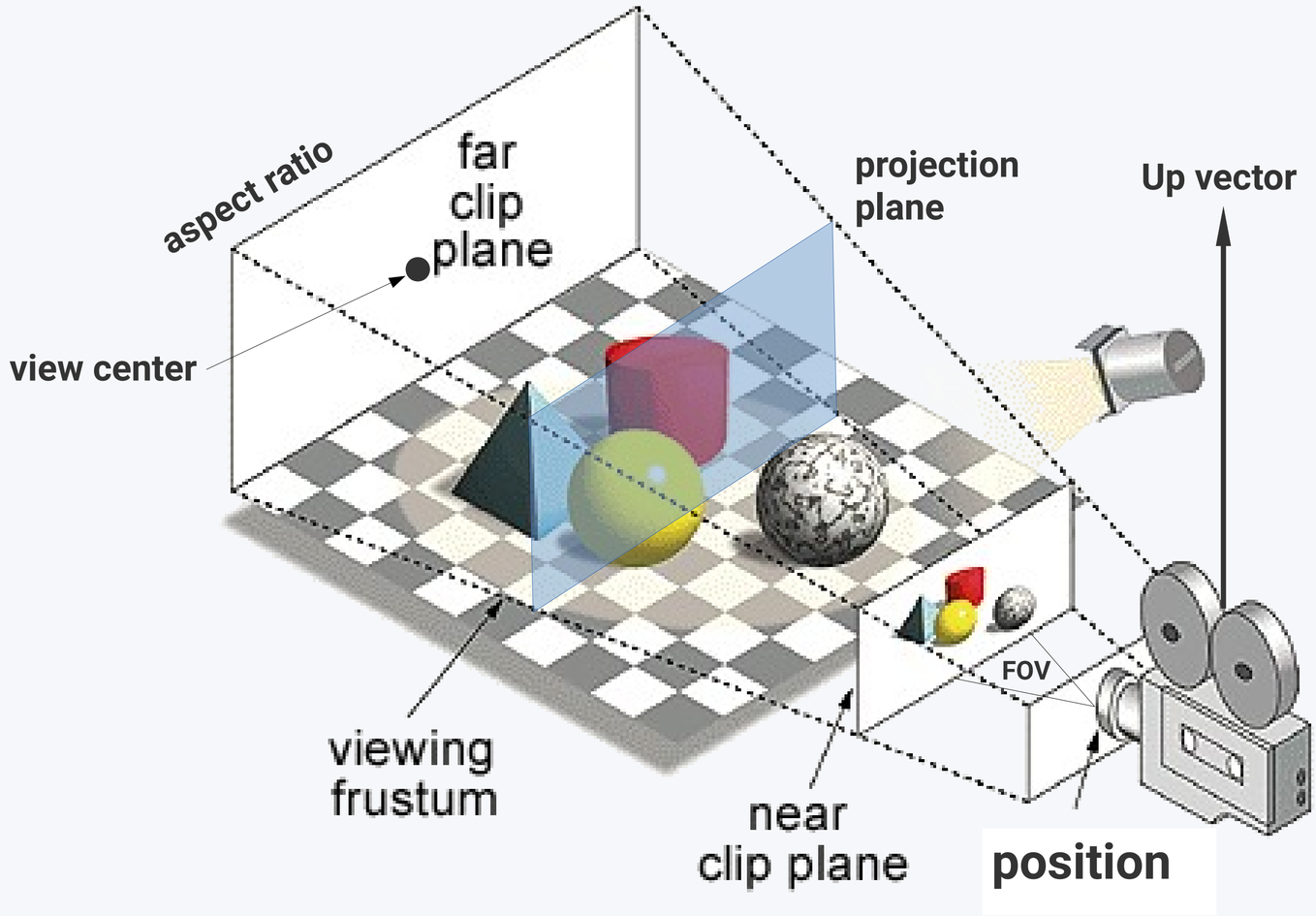
Transforma os vértices de todos os modelos da cena de modo que estes sejam vistos a partir da posição onde a câmera se encontra.



QtCon BR

Modelo de Câmera

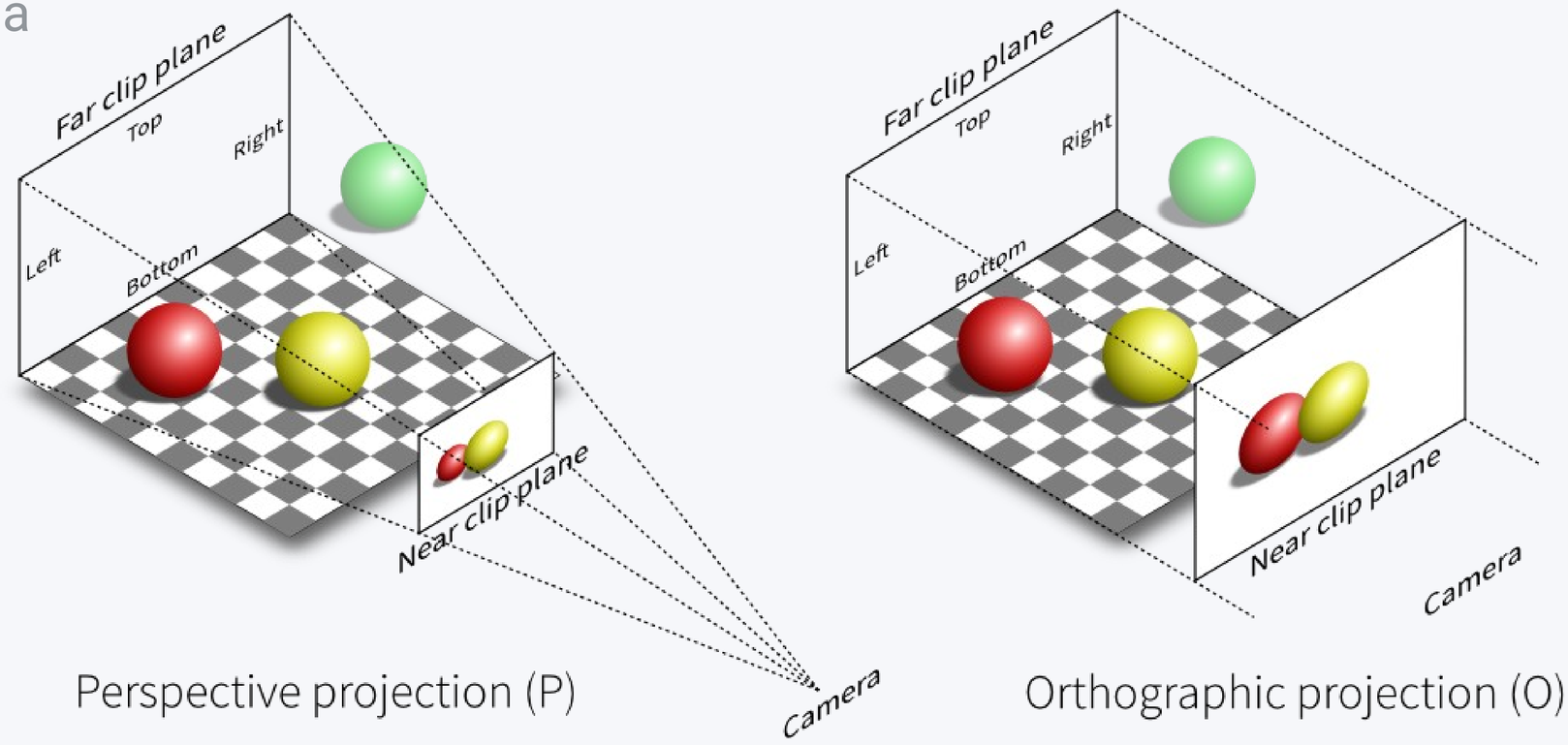
Transforma os vértices de todos os modelos da cena de modo que estes sejam vistos a partir da posição onde a câmera se encontra.



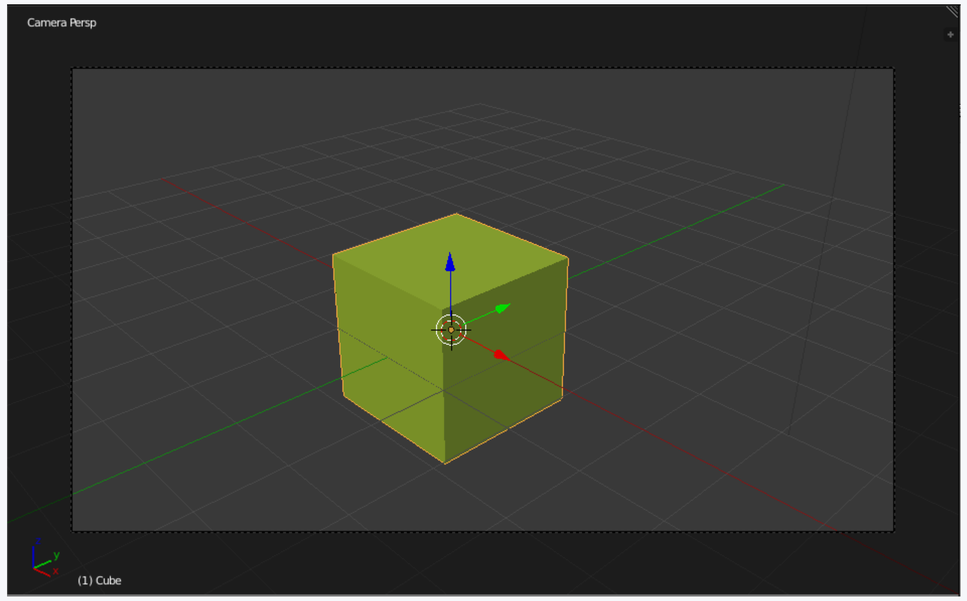
QtCon BR

Transformação de Projeção

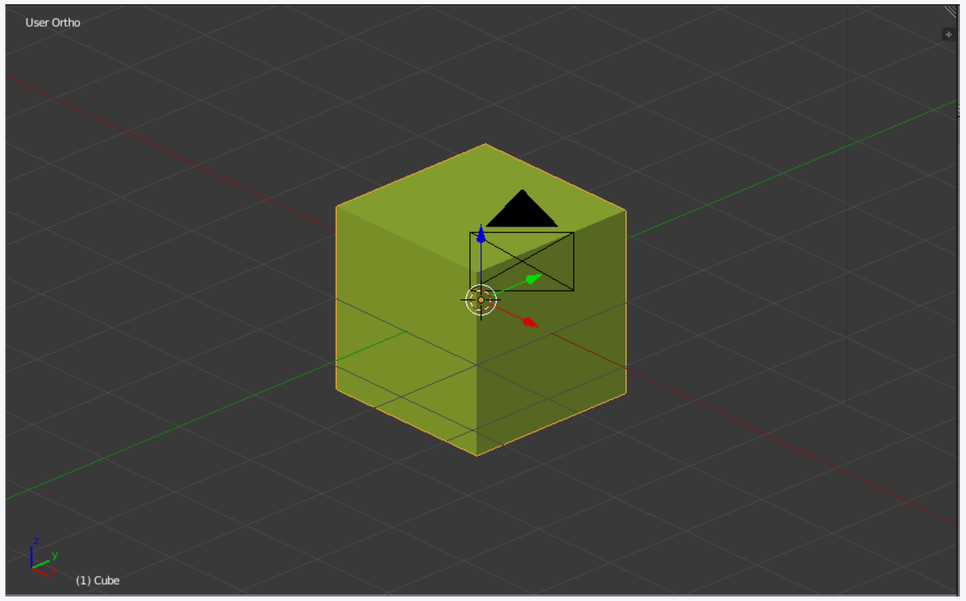
Projeta a cena do plano de projeção, descartando a coordenada Z.



Transformação de Projeção



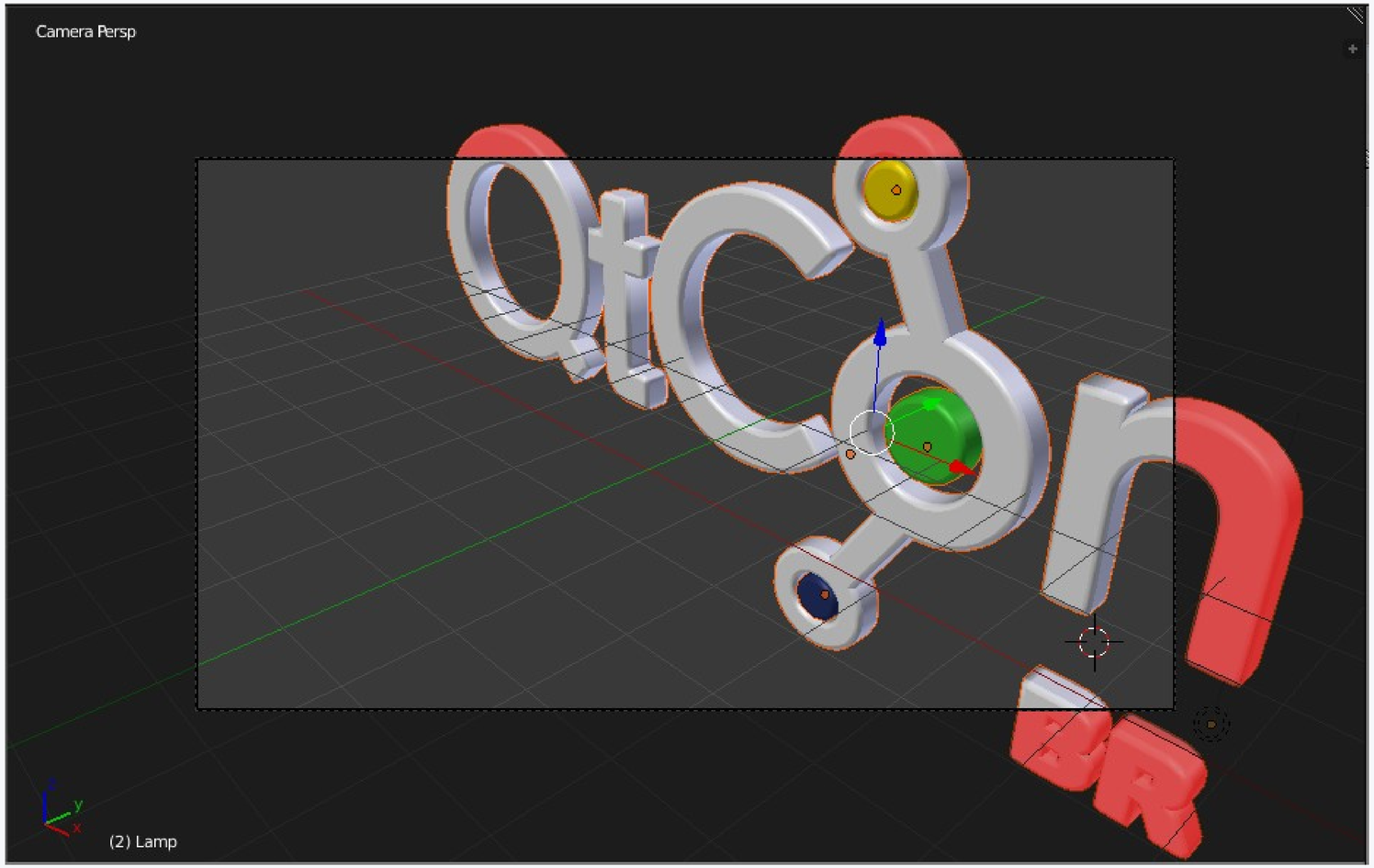
PERSPECTIVA



ORTOGRÁFICA

QtCon BR

Clipping

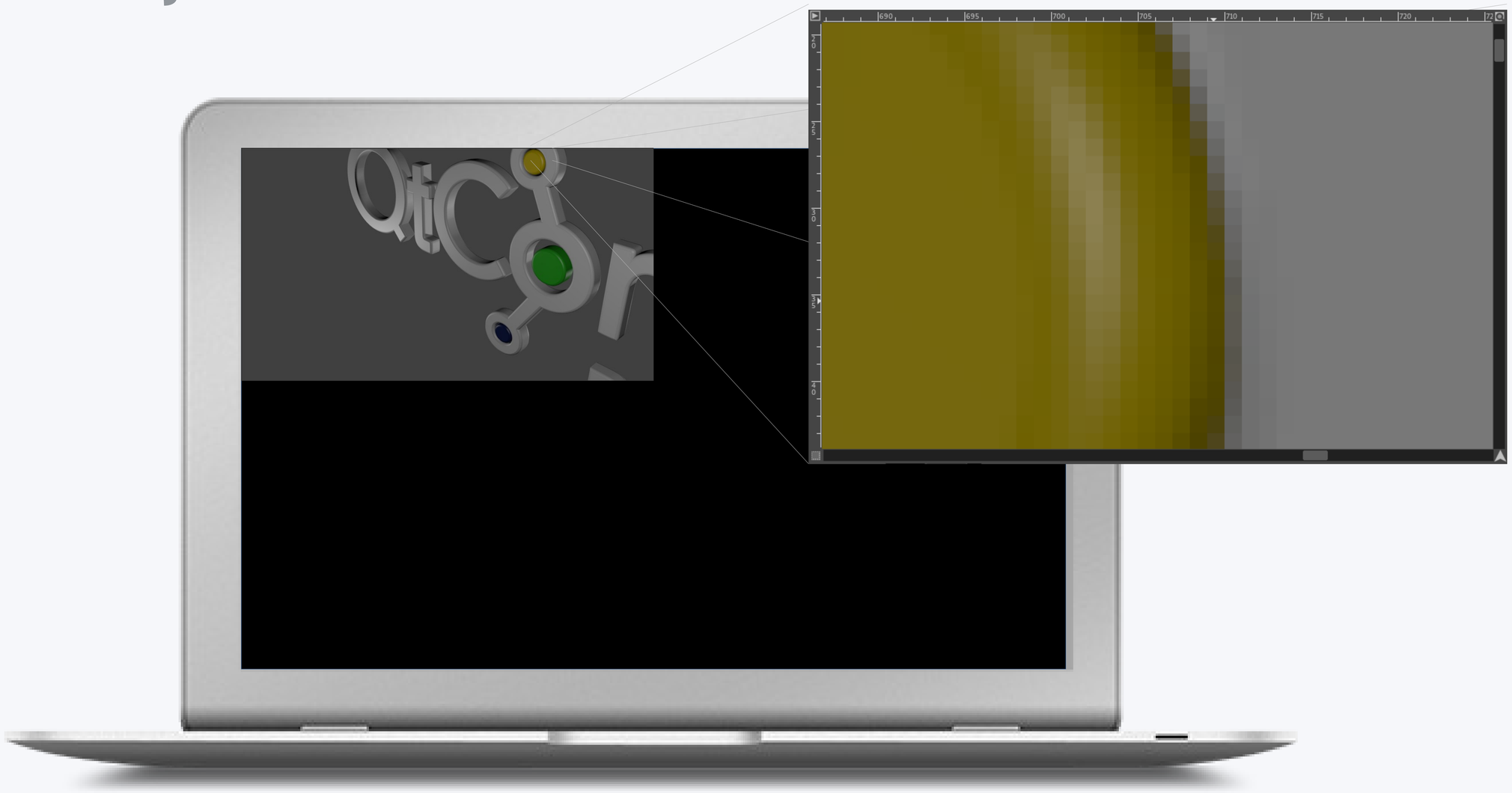


QtCon BR

Transformação para Viewport

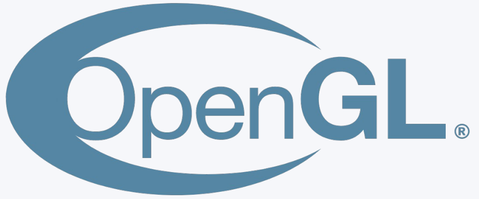
QtCon BR

Rasterização



QtCon BR

OpenGL



O que é o OpenGL?

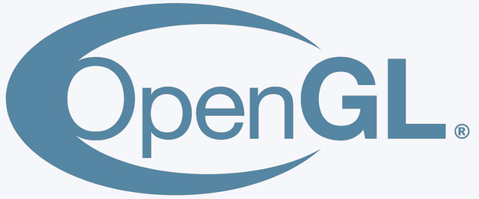
> É uma biblioteca multiplataforma para programação de GPUs modernas para realização de renderização 3D em tempo-real.

Foi criado no início da década de 1990 como uma padronização multiplataforma da biblioteca GL, desenvolvida pela Silicon Graphics Inc.

Utilizado em aplicações de CAD, realidade virtual, visualização científica, visualização de informação, simuladores de voo e jogos.

Em 2006, passou a ser mantigo pelo Khronos Group.

OpenGL



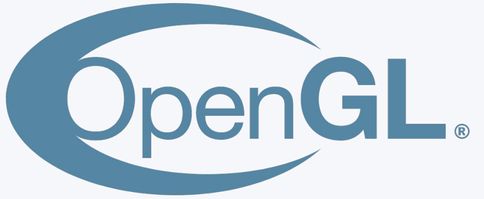
O Khronos Group também mantém variantes do OpenGL tais como o OpenGL ES (OpenGL for Embedded Systems) e o WebGL (binding JavaScript para renderização 3D em web browsers).

Algumas bibliotecas foram criadas para usar a GPU para aplicações de propósito geral. Exemplos incluem o CUDA (nVidia) e o OpenCL (Khronos Group).

Bibliotecas auxiliares para criação/gerenciamento de janelas OpenGL e gerenciamento de input:

- OpenGL Utility Library (GLU – *deprecated*)
- OpenGL Utility Toolkit (GLUT)
- freeglut
- GLFW
- GLEW (facilita o uso do OpenGL em diferentes versões e com diferentes extensões)

OpenGL



O Mesa 3D v18 implementa a especificação OpenGL v4.6 (atual) em software.

Vulkan

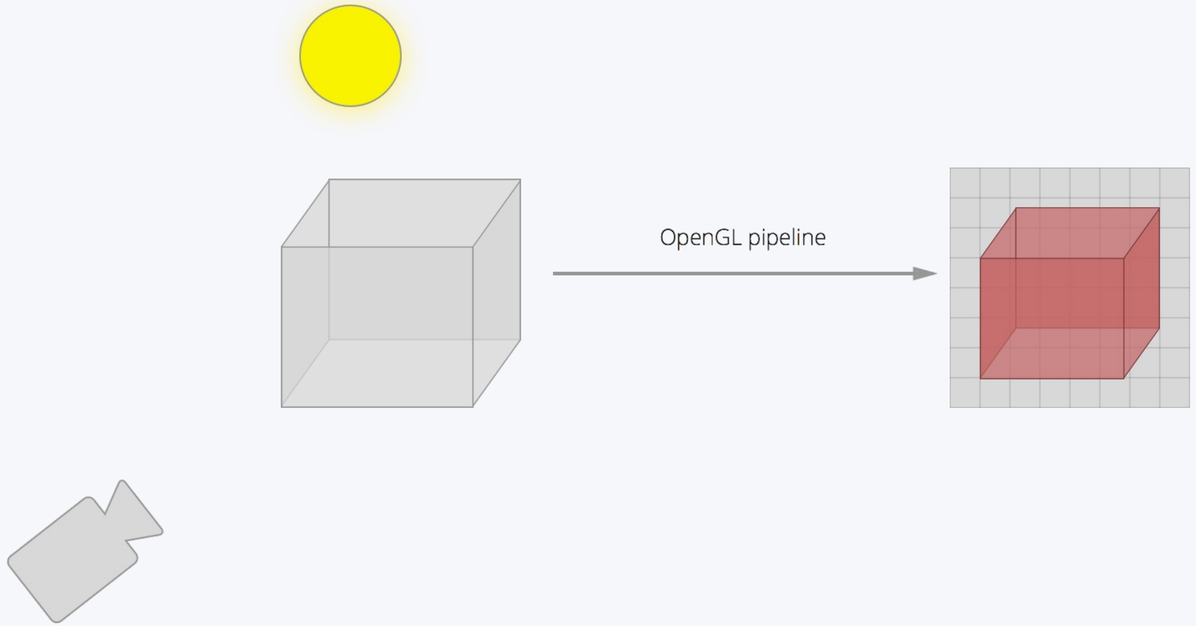
Nova API, mantida pelo Khronos Group, que unifica o OpenGL e o OpenGL ES e disponibiliza uma solução de alto desempenho e uso mais balanceado de tarefas entre múltiplas CPUs/GPUs.

Inicialmente chamado de “next generation OpenGL initiative” (OpenGL Next).

Primeira versão lançada em 16 de fevereiro de 2016.

Pipeline Gráfico no OpenGL

Modelo anterior (*deprecated*)

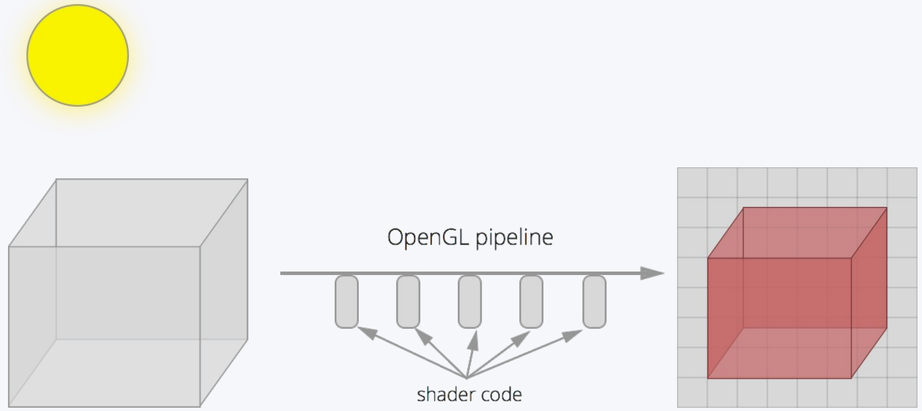


Pipeline Gráfico no OpenGL

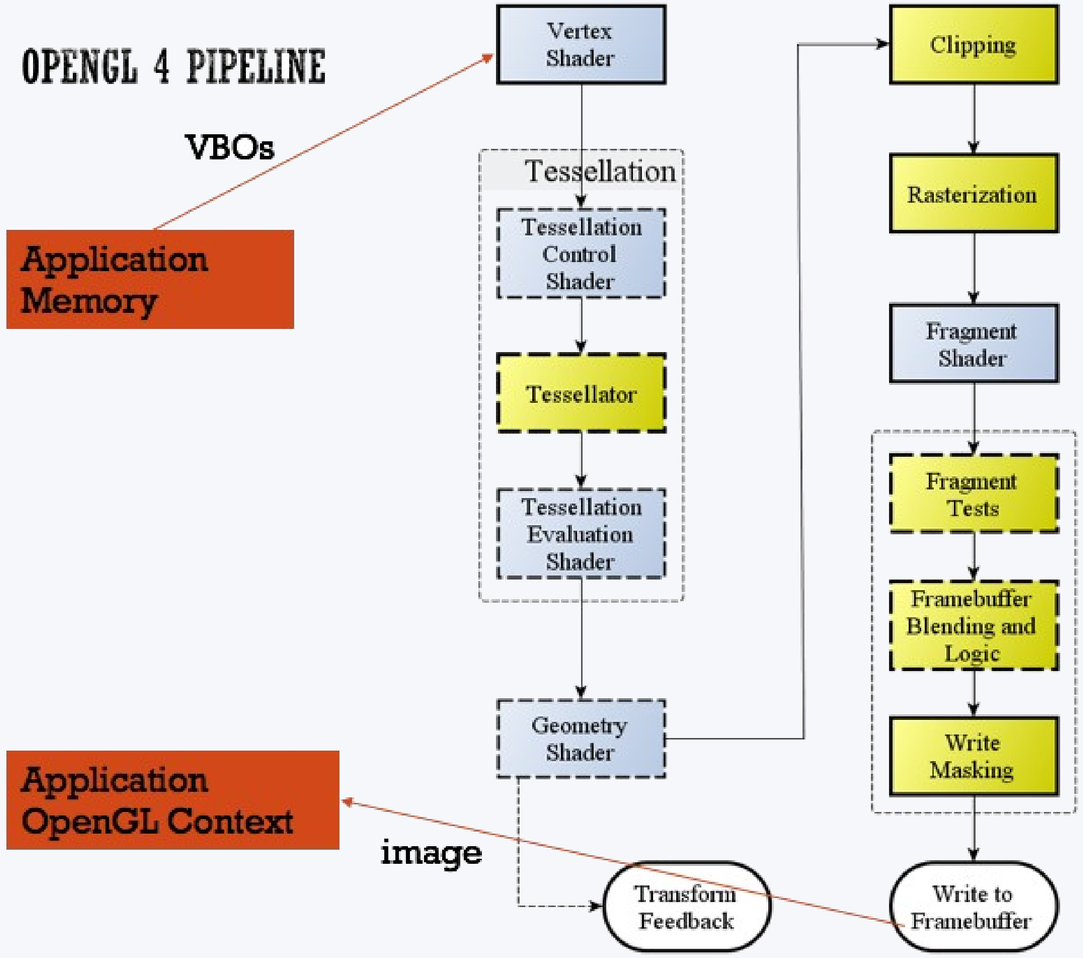
Modelo atual (*shaders*)

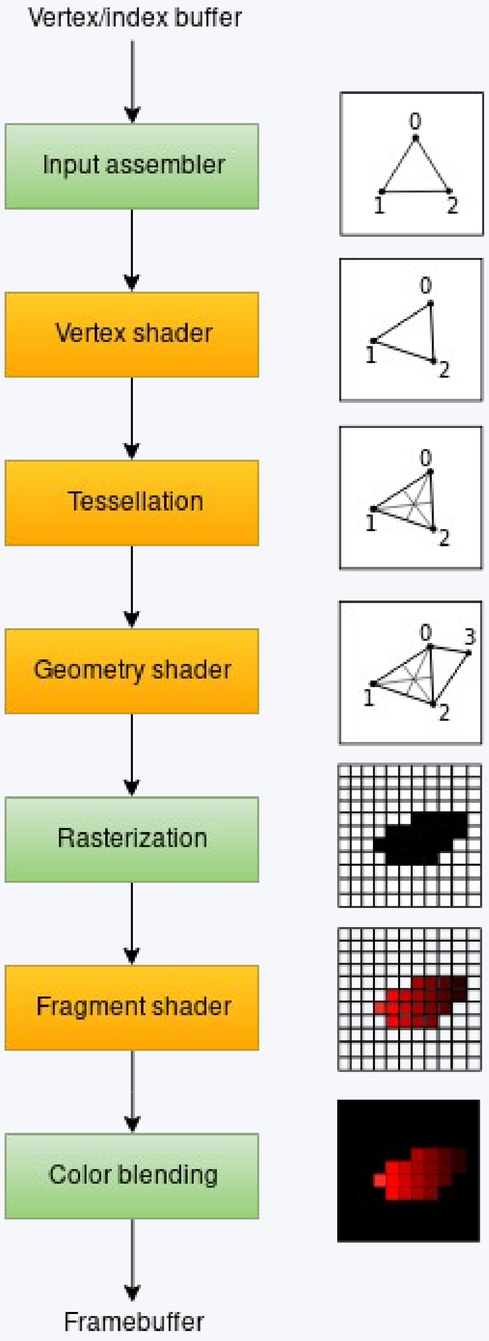
O que é um *shader*?

Tipo de programa de computador, originalmente usado para *shading* (cálculo de luz, escuridão e cor) em sistemas de CG, mas atualmente utilizado para realizar uma variedade de funções não relacionadas a *shading* ou mesmo não relacionadas a CG. São desenvolvidos utilizando a *OpenGL Shading Language* (GLSL).



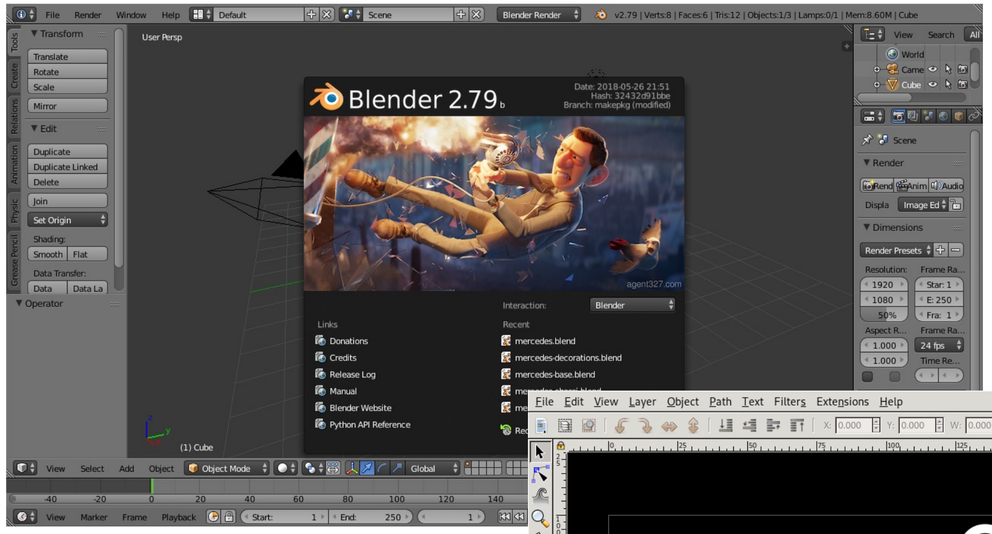
Pipeline Gráfico no OpenGL

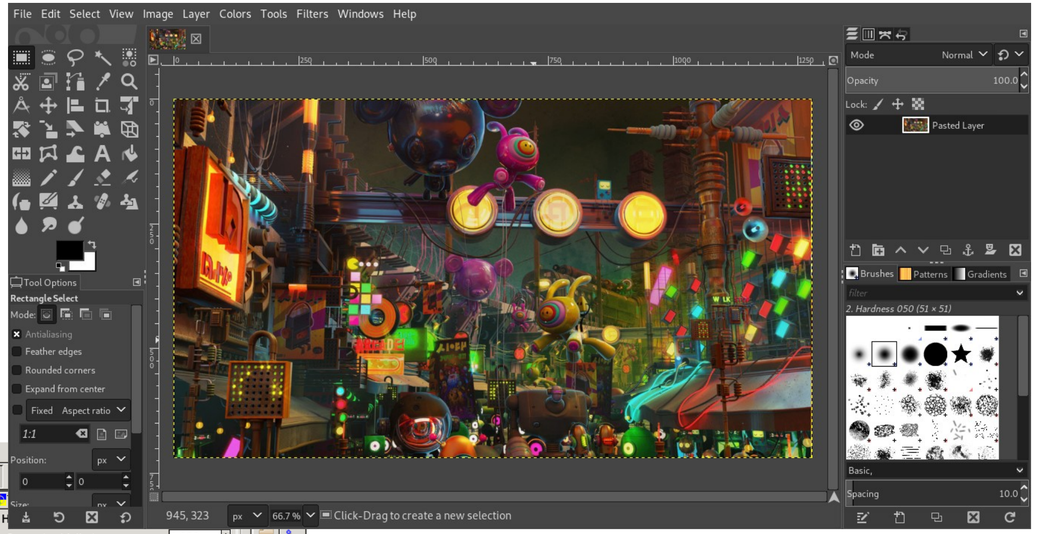


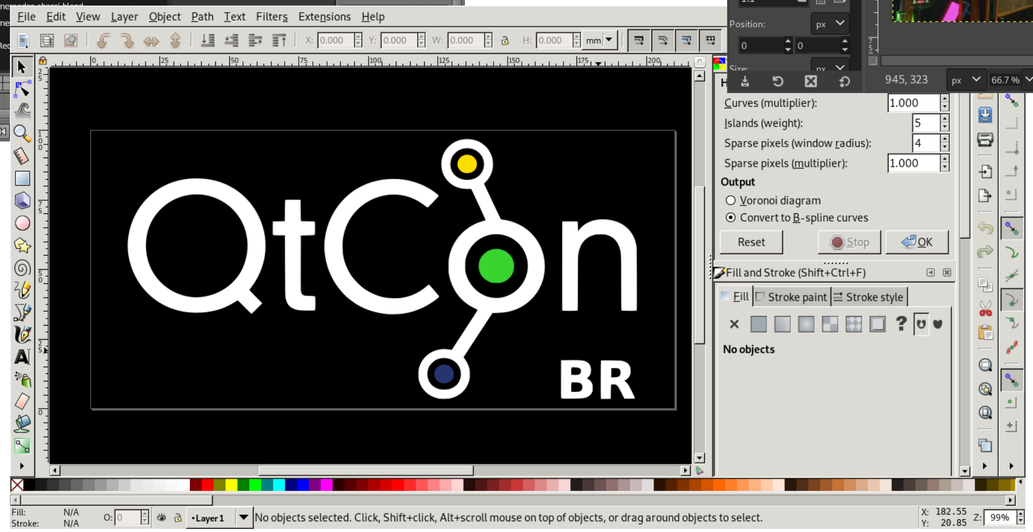


QtCon BR

Ferramentas Open Source para Computação Visual







O Qt

O Qt é um toolkit para desenvolvimento multiplataforma de aplicações em diversos domínios, com foco em execução nativa, excelente desempenho e produtividade.

Qt


BMW GROUP
Rolls-Royce Motor Cars Limited
bluesystems
technologies for a better world
Ableton
Nextcloud
OPERA software
vtk Visualization Toolkit
AUTODESK
Sun VirtualBox
skype
Google
last.fm
the social music revolution
KDAB
Adobe
MENDELEY
basysKom

O Qt

Por que usar?

- Tecnologia madura (desenvolvido há 23 anos).
- Alta produtividade (mesmo com C++, melhor ainda com QML/JS).
- Rico em funcionalidades (47 módulos, 1647 classes).
- Efetivo para desenvolvimento multiplataforma.
- Excelente documentação e comunidade ativa.
- Excelente desempenho (aceleração via hardware no QML).
- Diversas bibliotecas de terceiros (KF5, inqlude.org).
- Open Governance com licença dual (LGPL e comercial).

INTRODUÇÃO AO Qt3D

Objetivos, arquitetura, funcionalidades suportadas, APIs C++ e QML

QtCon BR

O Qt3D

- É um motor de simulação *soft real-time* de propósito múltiplo.
- Suporta múltiplos domínios (ex: IA, lógica, áudio, computação gráfica 3D, etc).
- Viabiliza a construção de sistemas complexos de simulação nas áreas de engenharia mecânica, robótica, física, jogos, dentre outras.
- O Qt3D nasceu no escritório da Nokia em Brisbane.
- 1o release em 11 de abril de 2012 (Qt 4.8.1).
- Pouco sucesso até 2014 quando a KDAB assume o desenvolvimento do Qt3D (Qt3D 2.0), lançado como TP no Qt 5.5 e como módulo estável no Qt 5.7.
- Disponibiliza APIs tanto em C++ quando em QML.

O Qt3D

Principais funcionalidades:

– Arquitetura flexível, extensível e escalável baseada no padrão Entity-Component-System (ECS).
– Carga facilitada de geometrias definidas pelo usuário (em formatos OBJ, STL e outros).
– Sistema poderoso para definição de materiais, efeitos e passagens de renderização.
– Configuração *data-driven* (sem uso de código C++) do renderizador (*framegraph*).
– Suporte a diversas técnicas de renderização (*forward*, *deferred*, *early z-fill*, *shadow mapping*, etc).
– Suporte a todos os estágios de *shaders* GLSL.
– Bom suporte ao uso de texturas e *render targets*, incluindo HDRI.
– Primitivas geométricas e materiais simples são disponibilizados *out-of-the-box*.
– Tratamento de input via teclado e controle simples de câmera via mouse.
– Integração com interfaces QtQuick 2.

Hello Qt3D (C++)

```
1.  int main(int argc, char *argv[])
2.  {
3.      QGuiApplication a(argc, argv);
4.      Qt3DExtras::Qt3DWindow view;
5.
6.      Qt3DCore::QEntity *scene = createScene();
7.
8.      Qt3DRender::QCamera *camera = view.camera();
9.      camera->setPosition(QVector3D(0, 0, 40.0f));
10.
11.     view.setRootEntity(scene);
12.     view.defaultFrameGraph()->setClearColor(QColor(0, 128, 255, 255));
13.     view.show();
14.
15.     return a.exec();
16. }
```

```
#include <QGuiApplication>
#include <Qt3DRender/QCamera>
#include <Qt3DExtras/Qt3DWindow>
#include <Qt3DExtras/QTorusMesh>
#include <Qt3DExtras/QPhongMaterial>
#include <Qt3DExtras/QForwardRenderer>
```

Hello Qt3D (C++)

```
1.  Qt3DCore::QEntity *createScene()
2.  {
3.      Qt3DCore::QEntity *rootEntity = new Qt3DCore::QEntity;
4.
5.      Qt3DExtras::QTorusMesh *torusMesh = new Qt3DExtras::QTorusMesh;
6.      torusMesh->setRadius(5);
7.      torusMesh->setMinorRadius(2);
8.      torusMesh->setRings(100);
9.      torusMesh->setSlices(40);
10.
11.     Qt3DExtras::QPhongMaterial *torusMaterial = new Qt3DExtras::QPhongMaterial;
12.     torusMaterial->setDiffuse("red");
13.
14.     Qt3DCore::QEntity *torusEntity = new Qt3DCore::QEntity(rootEntity);
15.     torusEntity->addComponent(torusMesh);
16.     torusEntity->addComponent(torusMaterial);
17.
18.     return rootEntity;
19. }
```

Hello Qt3D (QML)


```
1.  import QtQuick 2.11
2.  import Qt3D.Core 2.0
3.  import Qt3D.Render 2.0
4.  import Qt3D.Extras 2.0
5.
6.  Entity {
7.      components: [
8.          RenderSettings {
9.              activeFrameGraph: ForwardRenderer {
10.                 clearColor: Qt.rgba(0, 0.5, 1, 1)
11.                 camera: Camera {
12.                     position: Qt.vector3d(0.0,0.0,40.0)
13.                 }
14.             }
15.         }
16.     ]
17.     Entity {
18.         components: [
19.             TorusMesh {
20.                 radius: 5
21.                 minorRadius: 2
22.                 rings: 100
23.                 slices: 40
24.             },
25.             PhongMaterial {
26.                 diffuse: “red”
27.             }
28.         ]
29.     }
30. }
```

Hello Qt3D (QML)

```
1.  #include <QGuiApplication>
2.  #include <Qt3DQuickExtras/qt3dquickwindow.h>
3.
4.  int main(int argc, char *argv[])
5.  {
6.      QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
7.      QGuiApplication app(argc, argv);
8.
9.      Qt3DExtras::Quick::Qt3DQuickWindow view;
10.     view.setSource(QUrl("qrc:/main.qml"));
11.     view.show();
12.
13.     return app.exec();
14. }
```

O Qt3D – APIs

- As APIs C++ e QML são equivalentes e seguem a mesma nomenclatura: QTorusMesh / TorusMesh.
- O desempenho é equivalente (ambos renderizam via OpenGL). A versão QML acrescenta somente o *overhead* da execução do interpretador *JavaScript*.
- Módulos do Qt3D 5.11 (C++ e QML):
  - **Qt3DCore:** *framework* ECS, *thread pool* de tarefas, comunicação *backend*←→*frontend*.
  - **Qt3DInput:** input via teclado e mouse, mapeamento avançado de dispositivos de entrada.
  - **Qt3DLogic:** execução de código por *frame* renderizado.
  - **Qt3DRender:** framework de renderização 3D, subsistema de materiais, efeitos e técnicas.
  - **Qt3DAnimation (TP):** aspecto de animação, *keyframes*, *morphing*.
  - **Qt3DExtras (TP):** controladores de câmera, geometrias e materiais pré-definidos.
  - **Qt3DScene2D (TP):** permite renderizar uma cena Qt Quick 2 como uma textura.
  - **QtQuick.Scene3D (TP – somente QML):** integra uma cena Qt3D em uma aplicação Qt Quick 2.

O Qt3D – Arquitetura

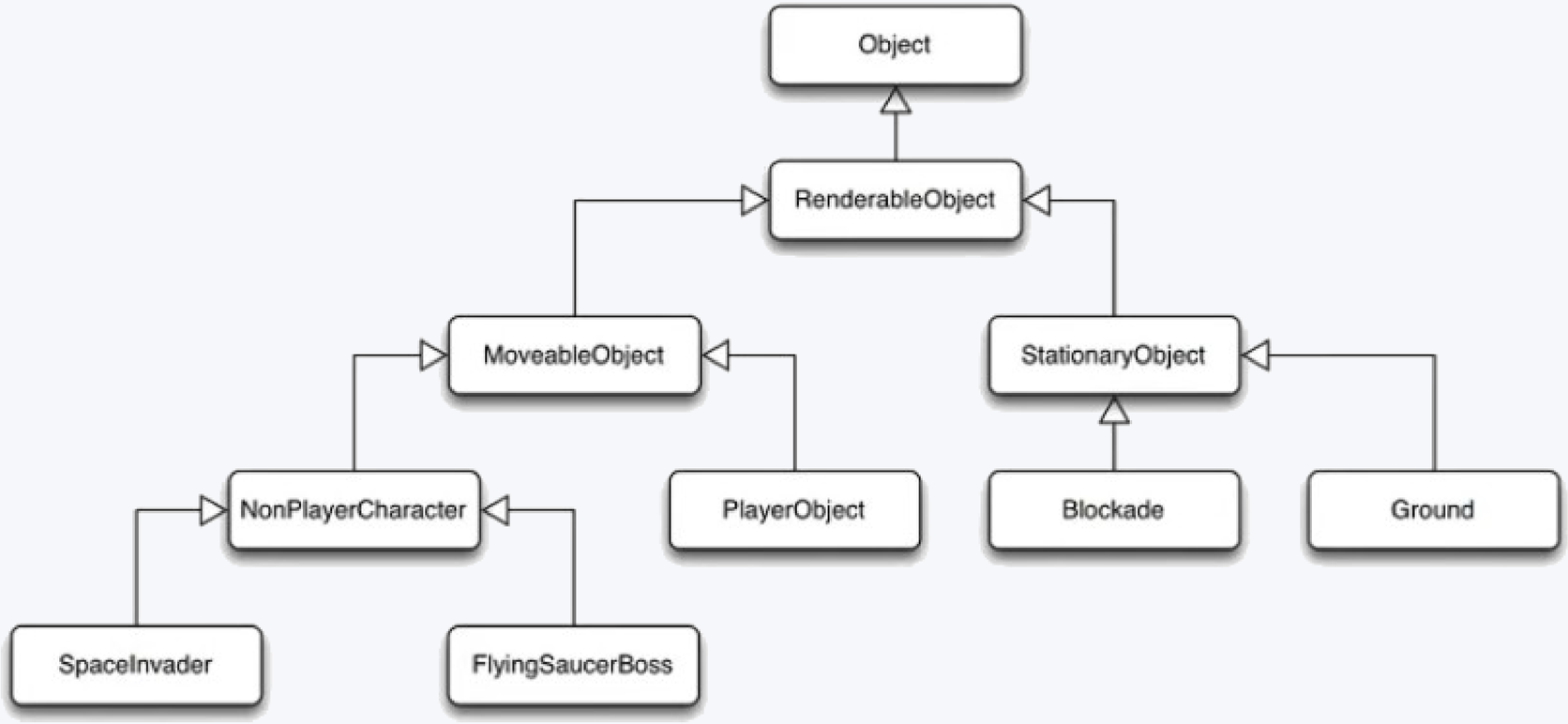
Desafios comuns no projeto de arquiteturas para jogos:

Itens com diferentes funcionalidades:

- Renderable
- Movable/Stationary
- Player/NonPlayer
- SoundEmitter
- PhysicsAnimated
- Collidable

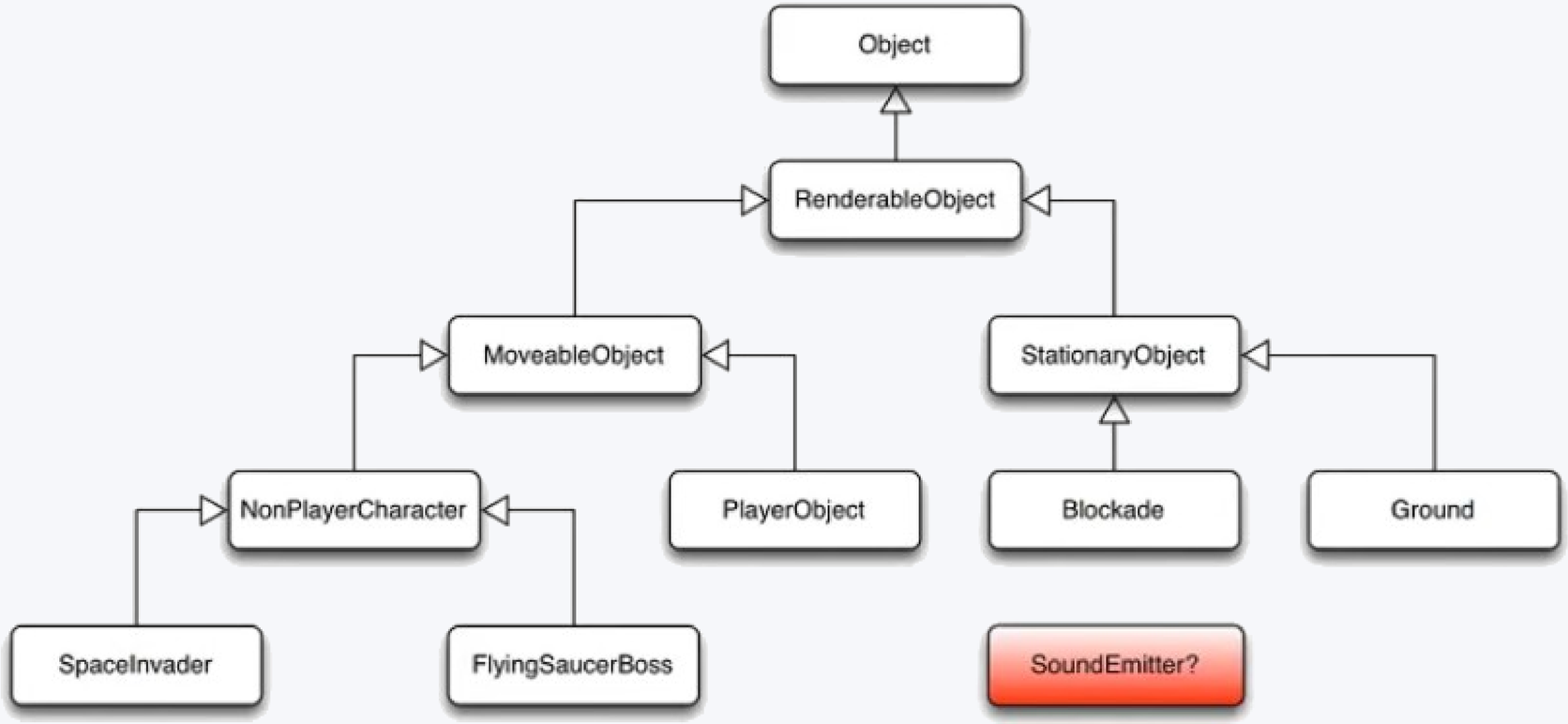
O Qt3D – Arquitetura

Soluções subótimas:



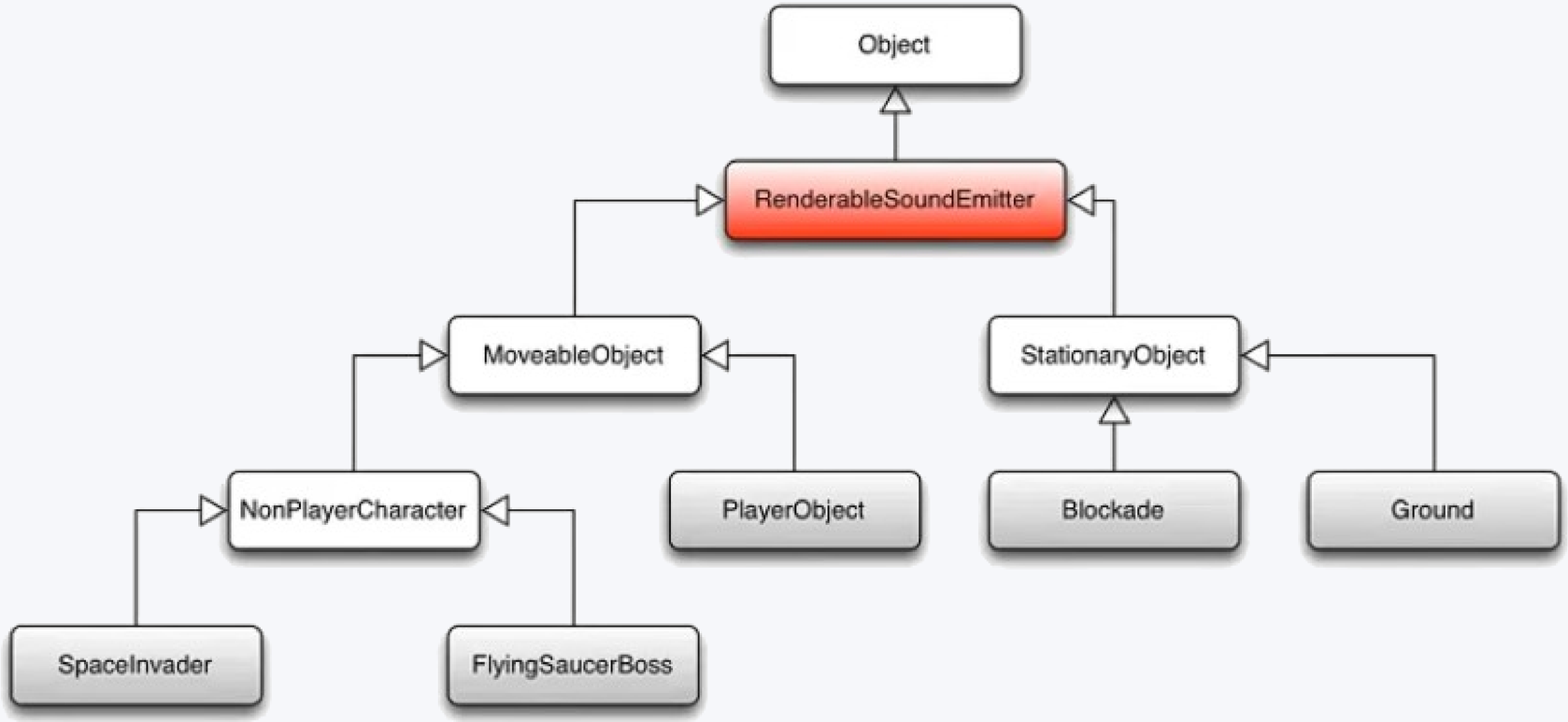
O Qt3D – Arquitetura

Soluções subótimas:



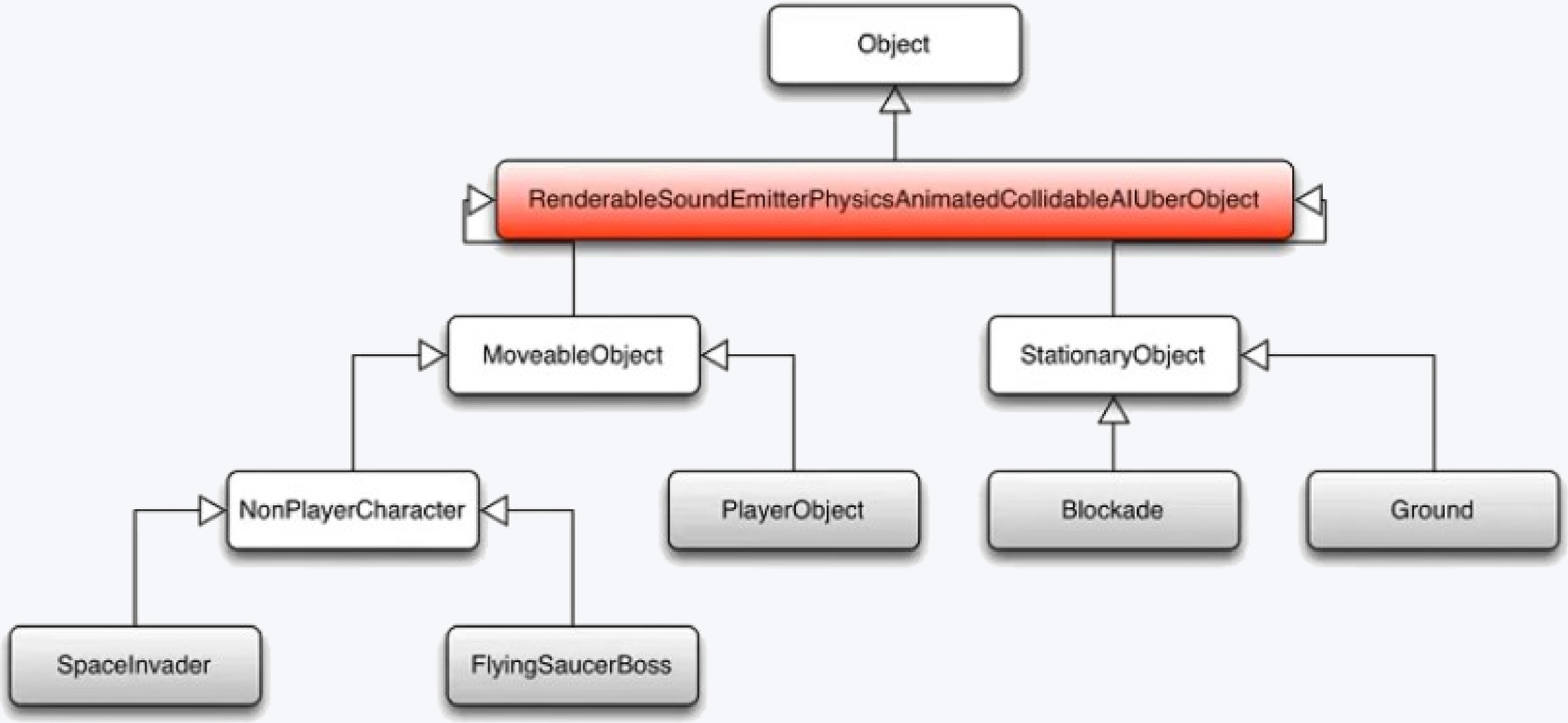
O Qt3D – Arquitetura

Soluções subótimas:



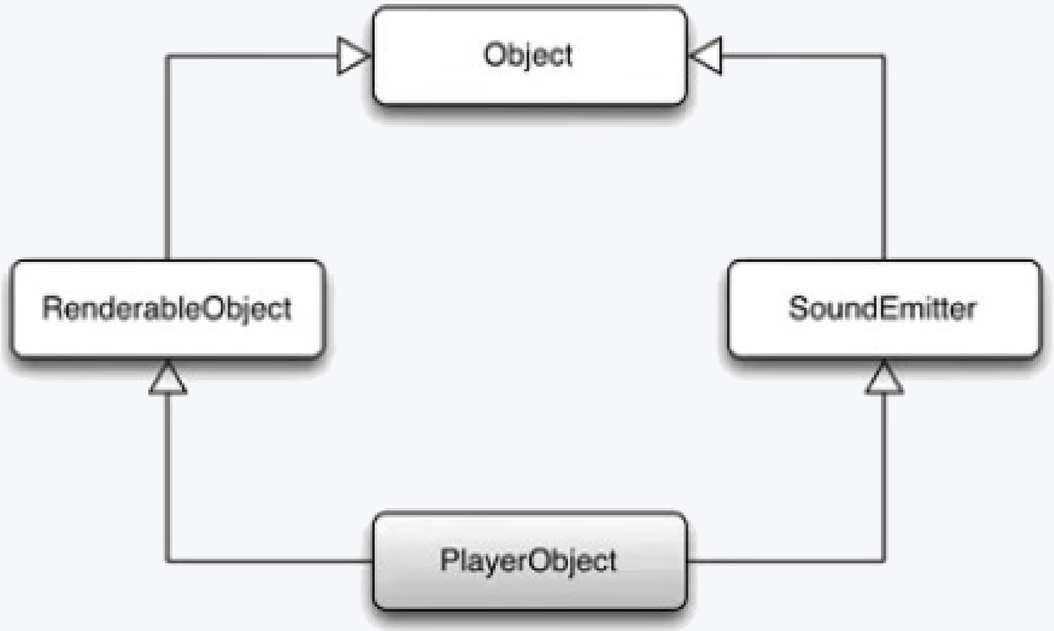
O Qt3D – Arquitetura

Soluções subótimas:



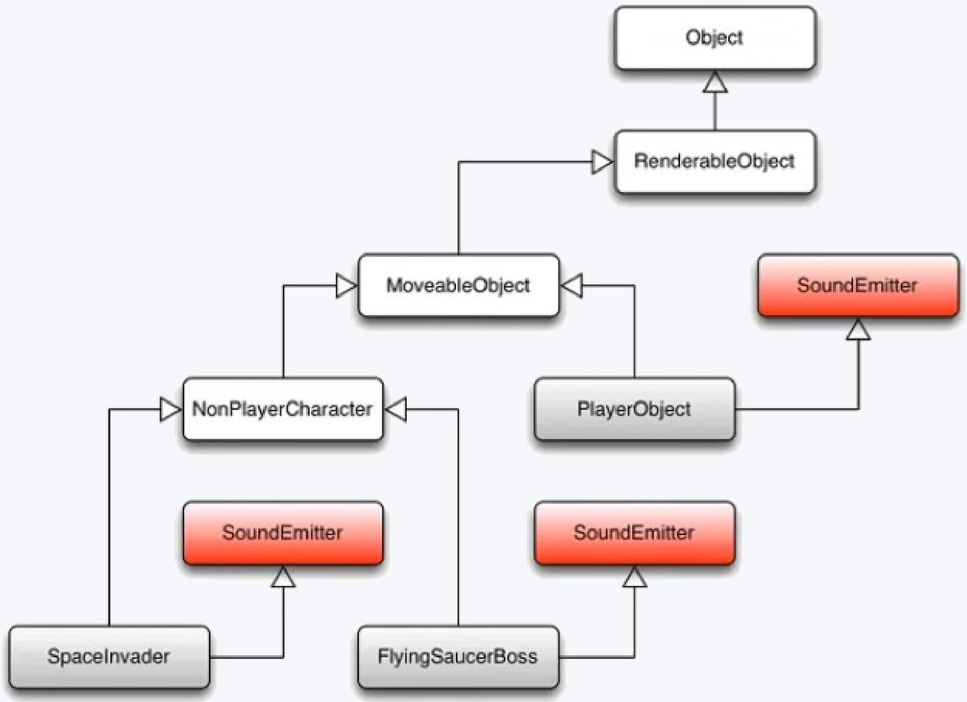
O Qt3D – Arquitetura

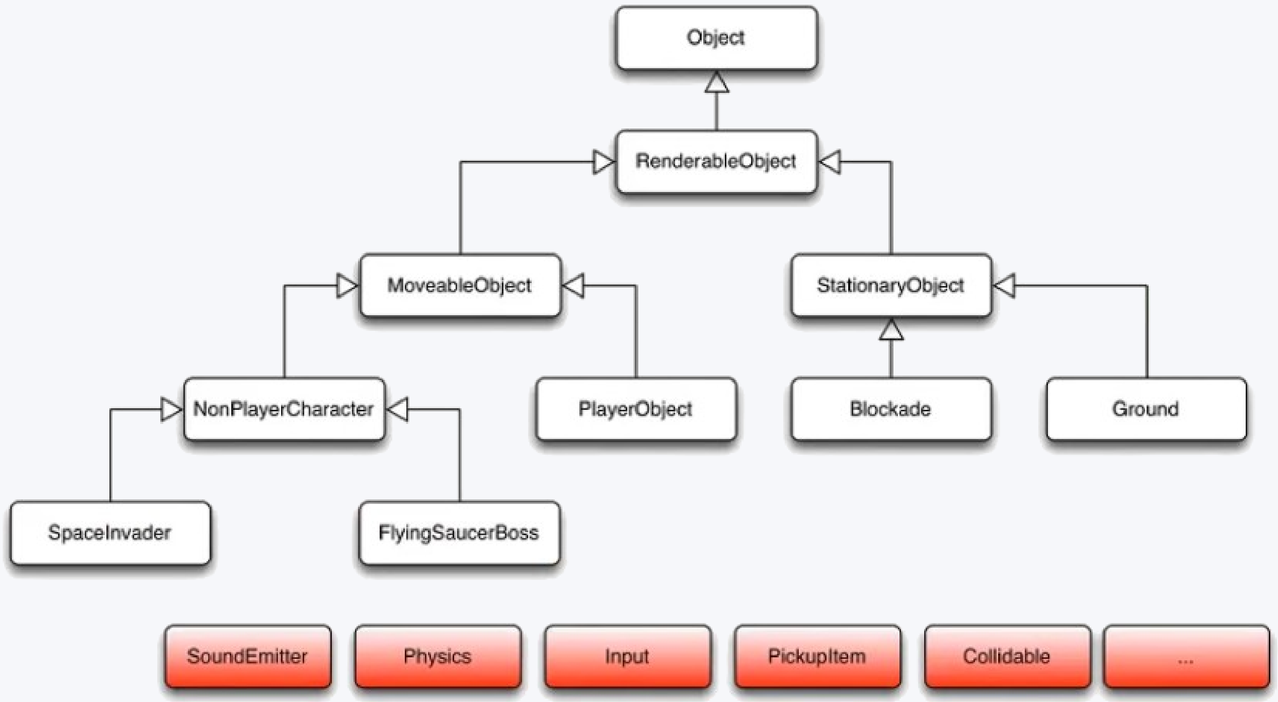
Soluções subótimas (herança múltipla):



O Qt3D – Arquitetura

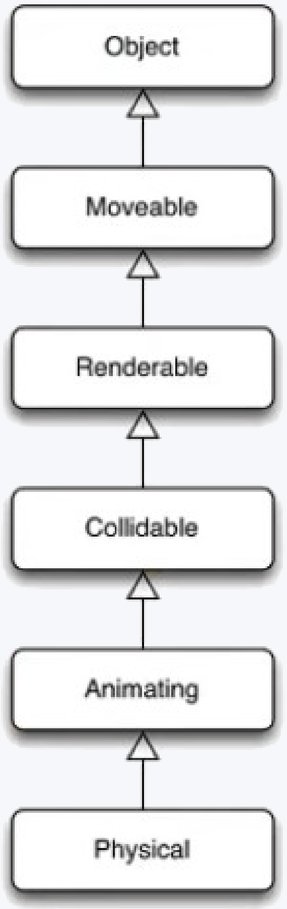
Soluções subótimas (herança *mix-in*):





O Qt3D – Arquitetura

Soluções subótimas (hierarquias lineares):



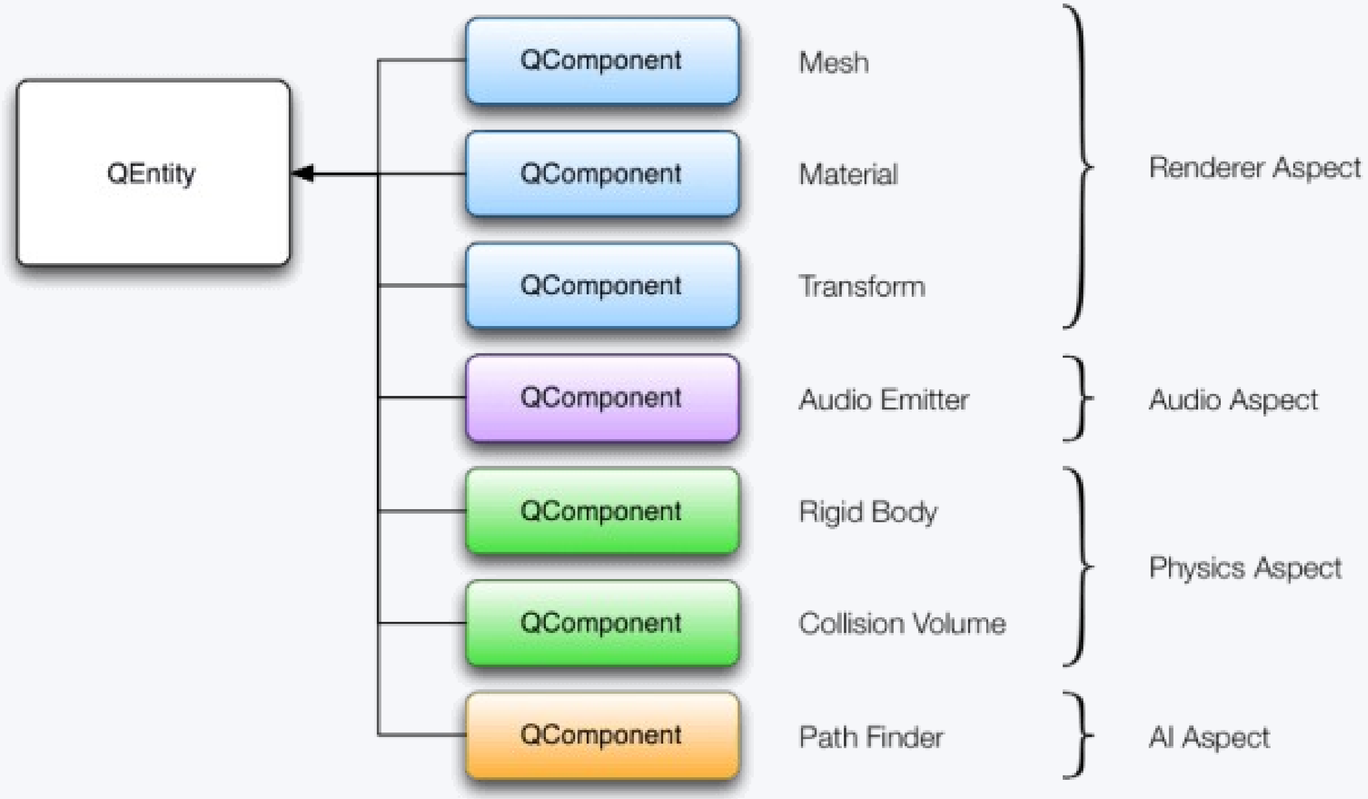
O Qt3D – Arquitetura

Entity-Component-System (ECS):

- Padrão arquitetural comum na área de jogos e que segue o princípio “prefira composição à herança”, trazendo mais flexibilidade e extensibilidade às aplicações.
- Participantes:
  - **Entity:** um objeto a ser simulado. Por si só, não possui comportamentos ou dados.
  - **Component:** objeto que agrega um novo comportamento ou dado a um *entity*.
  - **System:** uma *thread* de execução que itera sobre todos os *entities* que possuem componentes de interesse deste *system*. Cada *system* implementa uma funcionalidade particular integrada ao simulador. O Qt3D utiliza o termo ***aspect*** para se referir a um *system*.
- É extensível (novos *components* e *aspects* podem ser criados).
- É flexível em *run-time*.

O Qt3D – Arquitetura

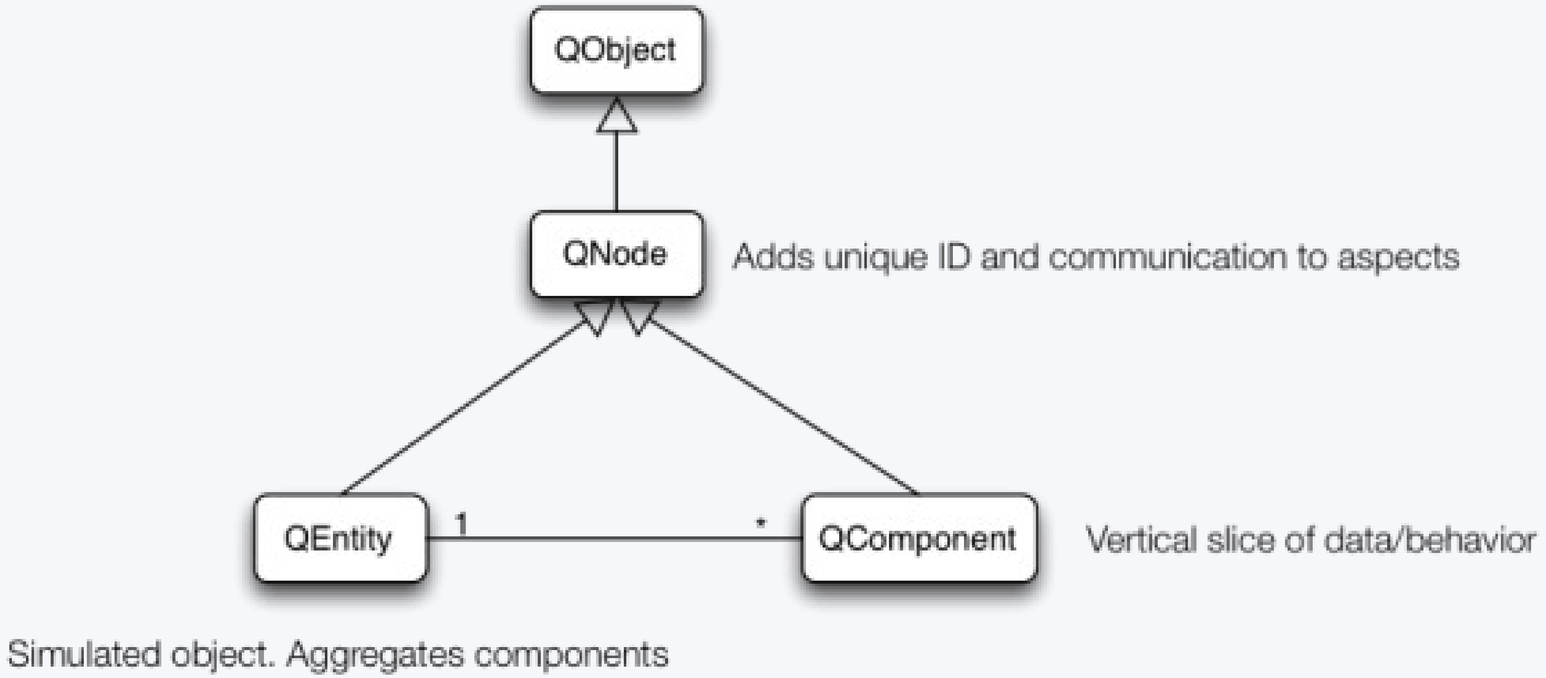
Entity-Component-System (ECS):



O Qt3D – Arquitetura

Entity-Component-System (ECS):

Implementação no Qt



O Qt3D – Arquitetura

Entity-Component-System (ECS):

O Qt3D disponibiliza atualmente os seguintes *aspects*:

- **QRenderAspect**
- **QInputAspect**
- **QLogicAspect**

  } automaticamente registrados

- **QAnimationAspect**

Um amplo conjunto de *components*:

- **QGeometryRenderer**, **QTransform**, **QInputSettings**, **QArmature**, **QAbstractLight**, **QMaterial**, **QObjectPicker**, **QRenderSettings**, dentre outros.

E cinco variantes de *entities*:

- **QEntity**, **QCamera**, **QSkyboxEntity**, **QabstractCameraController** e **QText2DEntity**.

GEOMETRIAS E TRANSFORMAÇÕES

Malhas poligonais primitivas, importando malhas, transformações

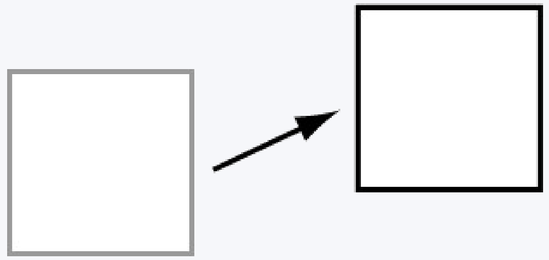
QtCon BR

Controladores e Parâmetros de Câmera

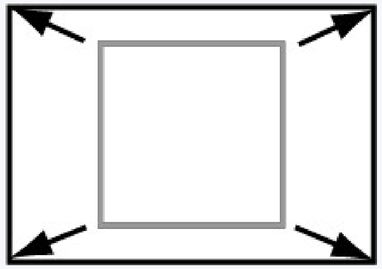
```
...
import Qt3D.Input 2.0

Entity {
    components: [
        RenderSettings {
            activeFrameGraph: ForwardRenderer {
                clearColor: Qt.rgba(0, 0.5, 1, 1)
                camera: camera
            }
        },
        InputSettings {}
    ]
```

```
    Camera {
        id: camera
        position: Qt.vector3d(0,0,40)
        viewCenter: Qt.vector3d(0,0,0)
        upVector: Qt.vector3d(0,1,0)
    }
    OrbitCameraController {
        camera: camera
    }
    ...
    ConeMesh {
        topRadius: 5
        bottomRadius: 10
        rings: 40
        slices: 100
        length: 10
    } ...
```

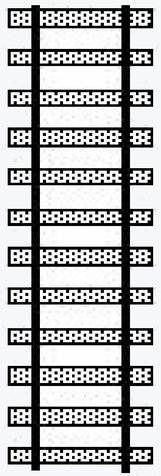
O Qt3D – Transformações

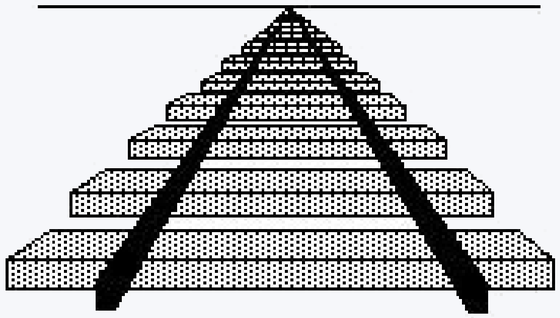


Translate

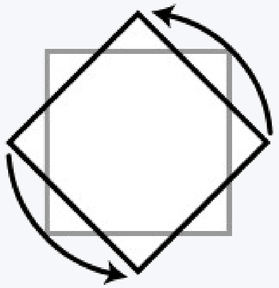


Scale

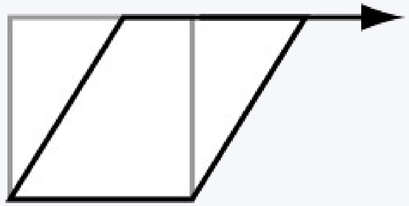




Perspective



Rotate



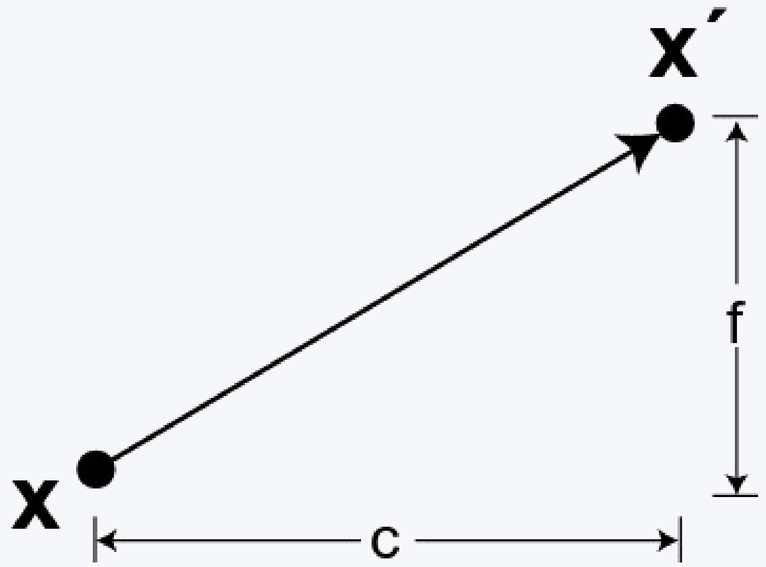
Shear

$$\mathbf{x}' = \begin{bmatrix} ax + by + c \\ dx + ey + f \end{bmatrix},$$

O Qt3D – Transformações

Translação pura:

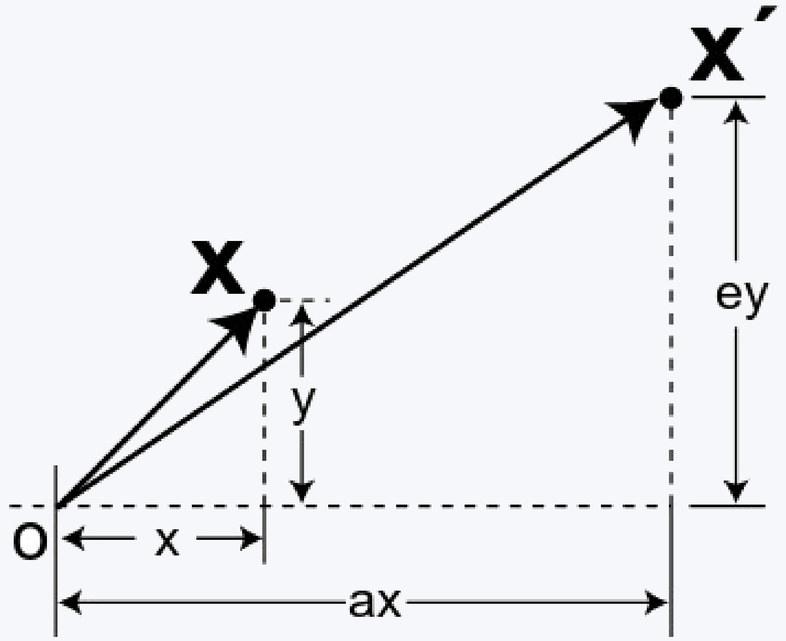
$$\mathbf{x}' = \begin{bmatrix} x + c \\ y + f \end{bmatrix}.$$



O Qt3D – Transformações

Escala pura:

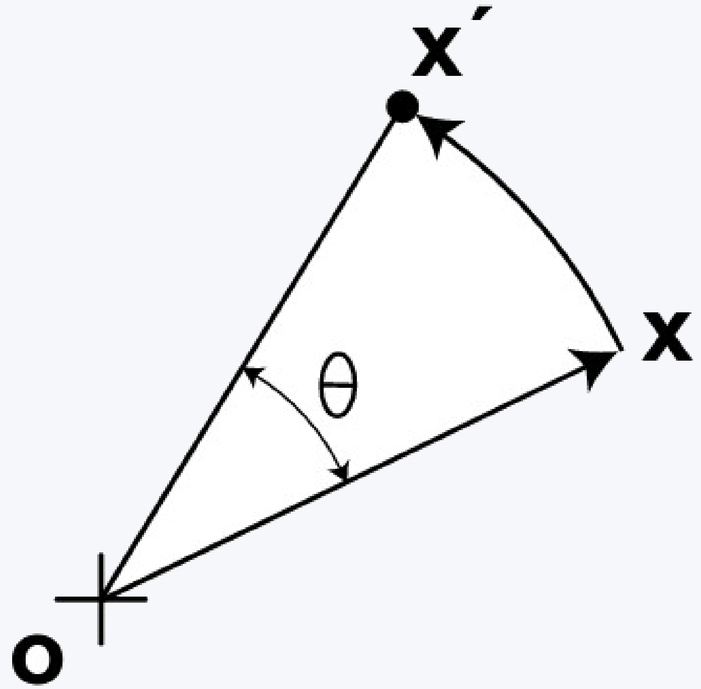
$$\mathbf{x}' = \begin{bmatrix} ax \\ ey \end{bmatrix}$$



O Qt3D – Transformações

Rotação em torno da origem:

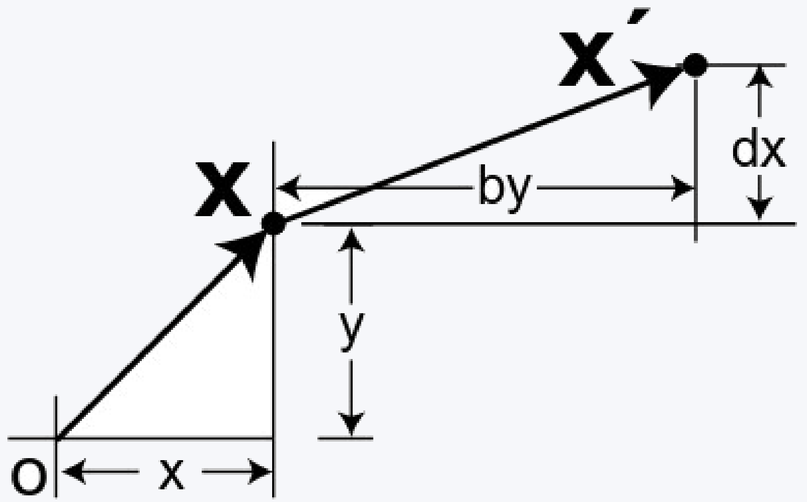
$$\mathbf{x}' = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}.$$



O Qt3D – Transformações

Cisalhamento (shear):

$$\mathbf{x}' = \begin{bmatrix} x + by \\ y + dx \end{bmatrix}.$$



O Qt3D – Transformações

Qt 3D

Representando transformações como matrizes:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ax + by \\ dx + ey \end{bmatrix} = \begin{bmatrix} a & b \\ d & e \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix},$$

$$\mathbf{x}' = R(H(S\mathbf{x}))$$

$$\mathbf{x}' = (RHS)\mathbf{x} = M\mathbf{x}$$

QtCon BR

O Qt3D – Transformações

Representando transformações como matrizes:

$$\text{Scale:} \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix}, \quad \text{Rotate:} \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \quad \text{Shear:} \begin{bmatrix} 1 & h_x \\ h_y & 1 \end{bmatrix},$$

Translação?

Transformação Perspectiva?

O Qt3D – Transformações

Coordenadas Homogêneas:

$$\begin{bmatrix} x \\ y \end{bmatrix} \Longrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

$$\text{Scale: } \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{Rotate: } \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}, \text{Shear: } \begin{bmatrix} 1 & h_x & 0 \\ h_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

$$\begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + c \\ dx + ey + f \\ 1 \end{bmatrix} \quad \Longrightarrow \quad \text{Translate: } \begin{bmatrix} 1 & 0 & \triangle x \\ 0 & 1 & \triangle y \\ 0 & 0 & 1 \end{bmatrix},$$

O Qt3D – Transformações

Coordenadas Homogêneas 3D: $\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Longrightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}.$

$$\text{Translate:} \begin{bmatrix} 1 & 0 & 0 & \triangle x \\ 0 & 1 & 0 & \triangle y \\ 0 & 0 & 1 & \triangle z \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \text{Scale:} \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\text{Shear:} \begin{bmatrix} 1 & h_{xy} & h_{xz} & 0 \\ h_{yx} & 1 & h_{yz} & 0 \\ h_{zx} & h_{zy} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

Transformação Perspectiva

$$\text{Rotation about the } x \text{ axis:} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x & 0 \\ 0 & \sin\theta_x & \cos\theta_x & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\text{Rotation about the } y \text{ axis:} \begin{bmatrix} \cos\theta_y & 0 & \sin\theta_y & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

$$\text{Rotation about the } z \text{ axis:} \begin{bmatrix} \cos\theta_z & -\sin\theta_z & 0 & 0 \\ \sin\theta_z & \cos\theta_z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

QtCon BR

O Qt3D – Transformações

O element Transform:

Define uma transformação contendo uma escala (via *vector3d*), uma rotação (via quatérnio) e uma translação (via *vector3d*). As operações são aplicadas nesta ordem.

Várias funções *helper* estão disponíveis:

– quaternion **fromAxesAndAngles**(vector3d axis1, real angle1, vector3d axis2, real angle2, vector3d axis3, real angle3)
– quaternion **fromAxesAndAngles**(vector3d axis1, real angle1, vector3d axis2, real angle2)
– quaternion **fromAxisAndAngle**(real x, real y, real z, real angle)
– quaternion **fromAxisAndAngle**(vector3d axis, real angle)
– quaternion **fromEulerAngles**(real pitch, real yaw, real roll)
– quaternion **fromEulerAngles**(vector3d eulerAngles)
– matrix4x4 **rotateAround**(vector3d point, real angle, vector3d axis)

Transformações são propagadas nas hierarquias de Entities.

Transformações

```
...
    Entity {
        components: [
            CuboidMesh {
                xExtent: 3
                yExtent: 1
                zExtent: 6
            },
            PhongMaterial { diffuse: "red" },
            Transform { id: transform }
        ]
    }
```

```
    SequentialAnimation {
        PropertyAnimation {
            target: transform; property: "rotationX"
            to: 45; duration: 1000
        }
        PropertyAnimation {
            target: transform; property: "rotationY"
            to: 90; duration: 1000
        }
        PropertyAnimation {
            target: transform; property: "rotationZ"
            to: 45; duration: 1000
        }
        running: true
    }
```

O Qt3D – Luzes

O Qt3D disponibiliza três tipos diferentes de luz:

- **PointLight:** luz pontual e que irradia energia de forma atenuada (à medida em que se afasta da fonte) em todas as direções.
- **DirectionalLight:** luz planar, localizada no infinito e que irradia energia de forma homogênea em uma única direção.
- **SpotLight:** luz cônica e que irradia energia de forma atenuada (à medida em que se afasta do eixo central vertical do cone) em uma única direção.

Luz Pontual Animada

```
Entity {
    components: [
        PlaneMesh { width: 20; height: 20 },
        PhongMaterial { diffuse: "red" }
    ]
}
Entity {
    components: [
        SphereMesh {
            radius: 2; slices: 40; rings: 40
        },
        PhongMaterial { diffuse: "yellow" },
        Transform {
            translation: Qt.vector3d(0, 2, 0)
        }
    ]
}
```

```
Entity {
    components: [
        SphereMesh { radius: 0.25 },
        Transform {
            translation: Qt.vector3d(-5, 5, 0)
            SequentialAnimation on translation {
                loops: Animation.Infinite
                Vector3dAnimation {
                    from: Qt.vector3d(-5, 5, 0)
                    to: Qt.vector3d(5, 5, 0)
                    duration: 5000
                }
                Vector3dAnimation { ... }
            }
        },
        PhongMaterial { diffuse: "blue" },
        PointLight { color: "blue"; intensity: 2 }
    ]
}
```

Luzes Direcional e Spot Animada

```
Entity {
    components: [
        DirectionalLight {
            worldDirection: Qt.vector3d(-1, -1, 0)
        }
    ]
}
Entity {
    components: [
        SphereMesh { radius: 0.25 },
        SpotLight {
            localDirection: Qt.vector3d(0, -1, -1)
            intensity: 1.0
            color: "white"
            CutOffAngle: 15
            SequentialAnimation on localDirection {
                loops: Animation.Infinite
                Vector3dAnimation {
                    from: Qt.vector3d(0, -1, -2)
                    to: Qt.vector3d(0, -1, 2)
                    Duration: 1000
                }
                Vector3dAnimation {
                    from: Qt.vector3d(0, -1, 2)
                    to: Qt.vector3d(0, -1, -2)
                    Duration: 1000
                }
            }
        },
        Transform {
            translation: Qt.vector3d(0, 5, 5)
        },
        PhongMaterial { diffuse: "white" }
    ]
}
```

O Qt3D – Carregando Malhas Externas

```
1. Entity {
2.     components: [
3.         Mesh { source: "suzanne.obj" },
4.         PhongMaterial { diffuse: "yellow" },
5.         Transform { translation: Qt.vector3d(0, 2, 0) }
6.     ]
7. }
```

Formatos suportados:

- Wavefront OBJ
- Stanford Triangle Format PLY
- STL (StereoLithography)
- Autodesk FBX

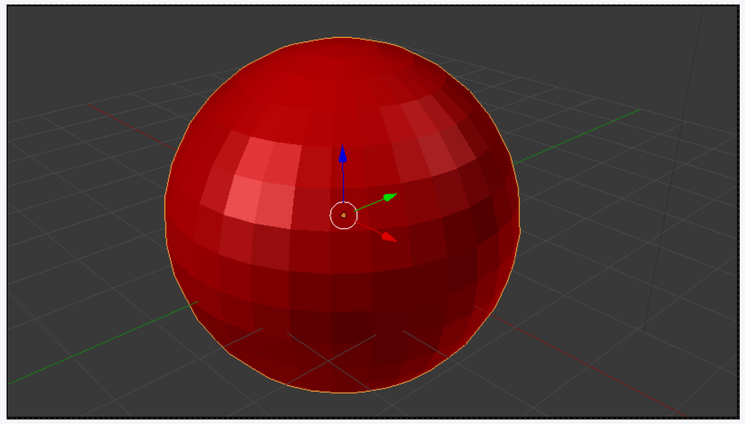
RENDERIZAÇÃO E ANIMAÇÕES

Materiais, luzes, PBR, animações

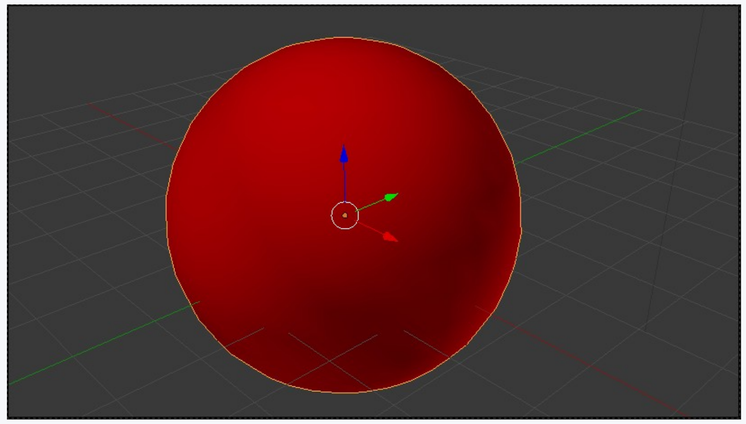
QtCon BR

O Qt3D – Materiais

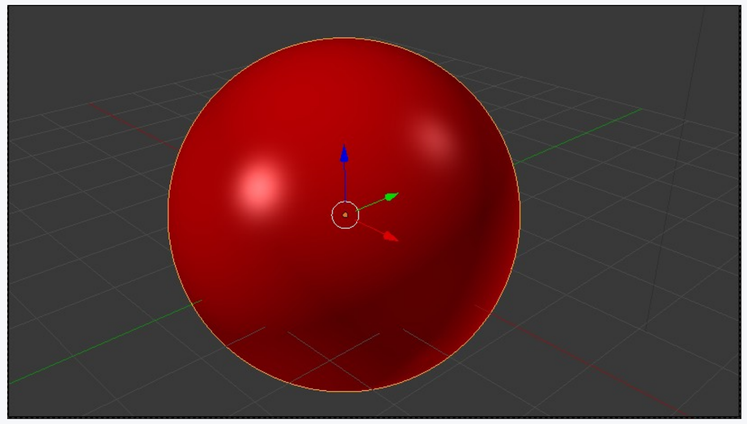
Materiais definem a aparência de uma superfície com base nas suas propriedades de reflexão da luz.
Envolvem a definição de técnicas de *shading*, mapeamentos, propriedades físicas e de áudio.
Principais técnicas de *shading*:



FLAT



GOURAUD

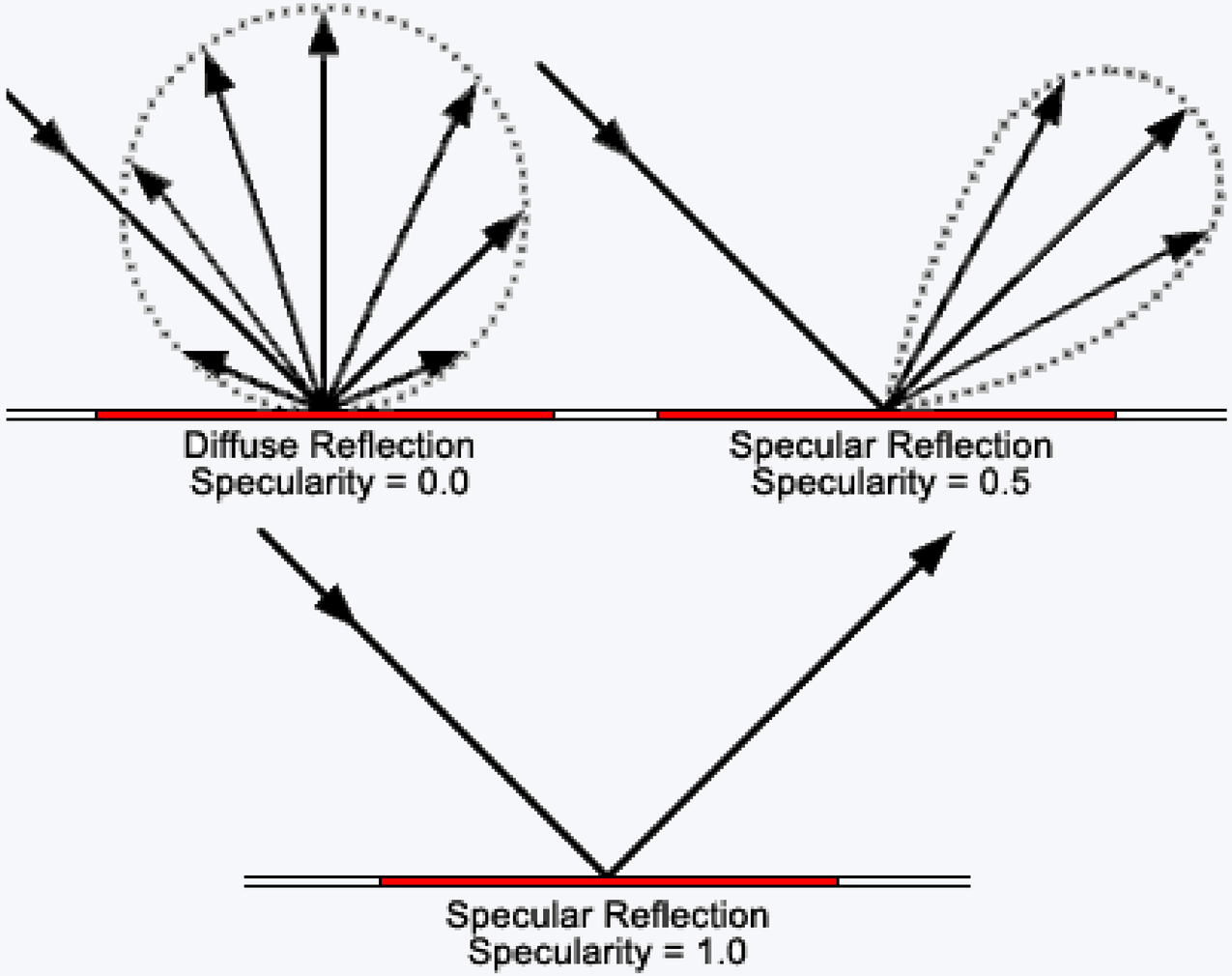


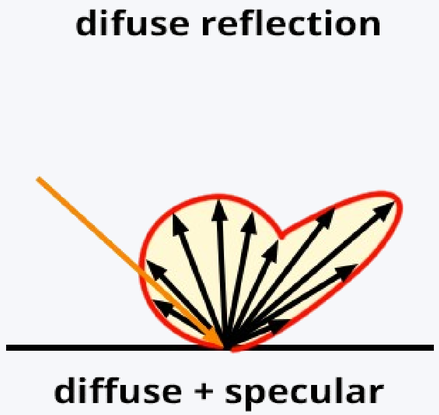
PHONG

O Qt3D – Materiais

Tipos de reflexão de luz:

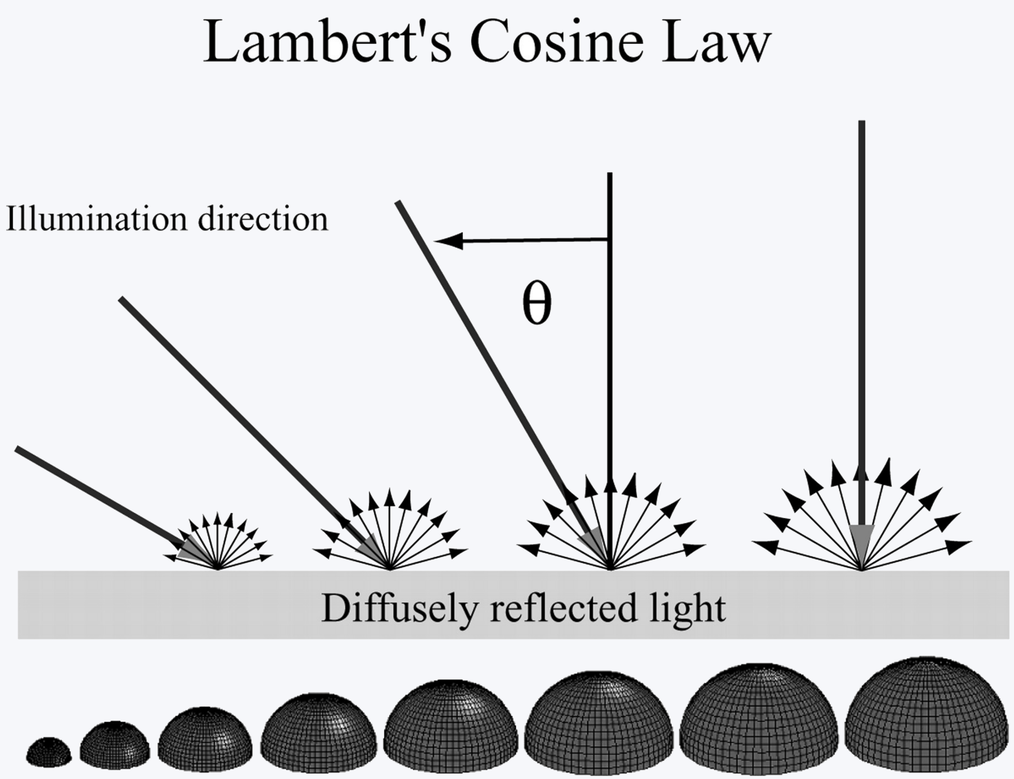
- **Difusa:** típica de objetos com aparência fosca. Ex: argila, papel, etc.
- **Especular:** típica de objetos brilhantes e com spots de reflexão. Ex: metais.
- **Ambiente:** emitida naturalmente na ausência de luzes.





O Qt3D – Materiais

Reflexão difusa:

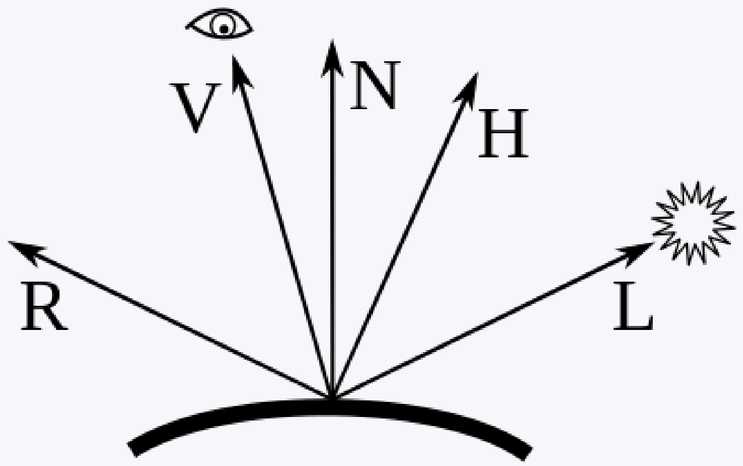


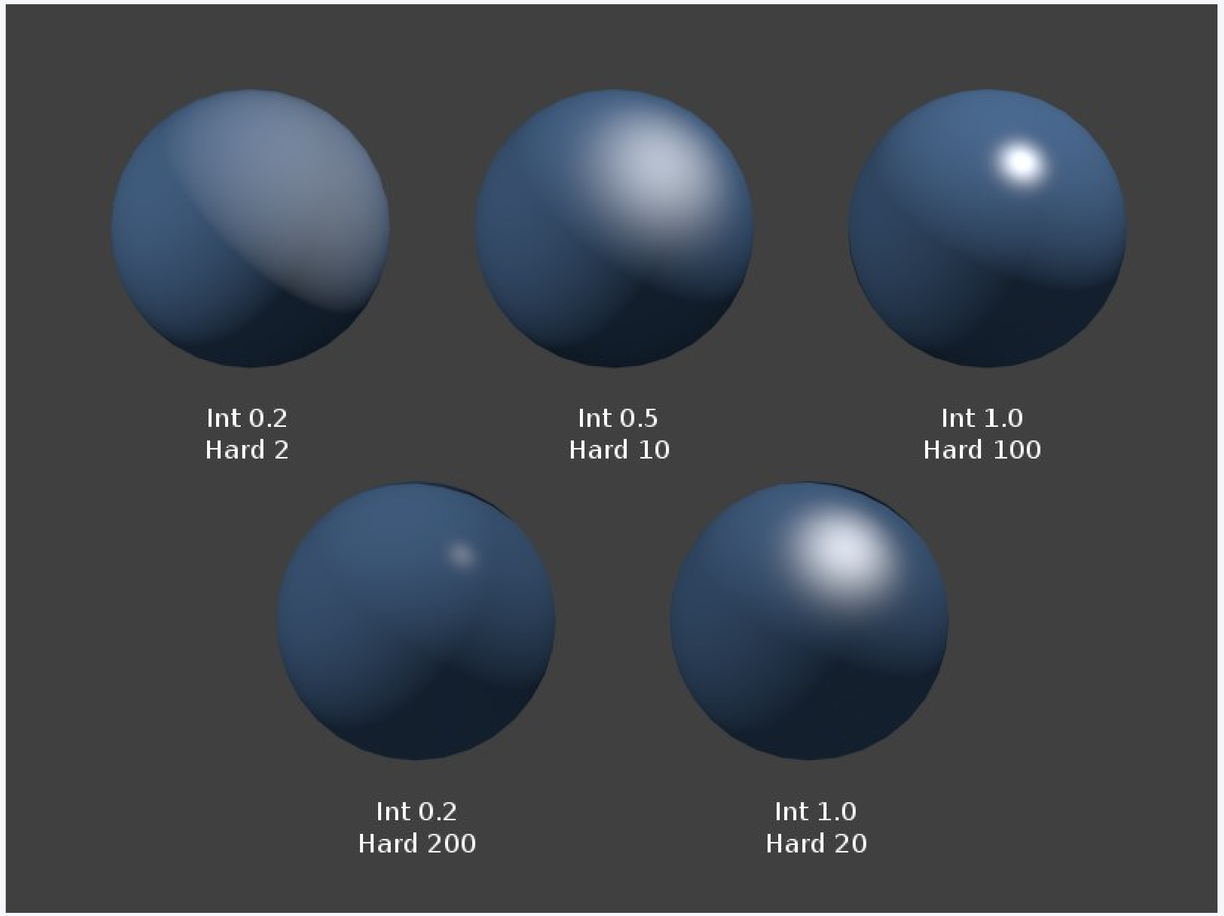
Angular distribution of reflected light is independent of illumination angle for a Lambertian reflector.

O Qt3D – Materiais

Reflexão especular (Phong):

$$k_\mathrm{s}(\hat{R}_m \cdot \hat{V})^\alpha i_{m,\mathrm{s}}$$





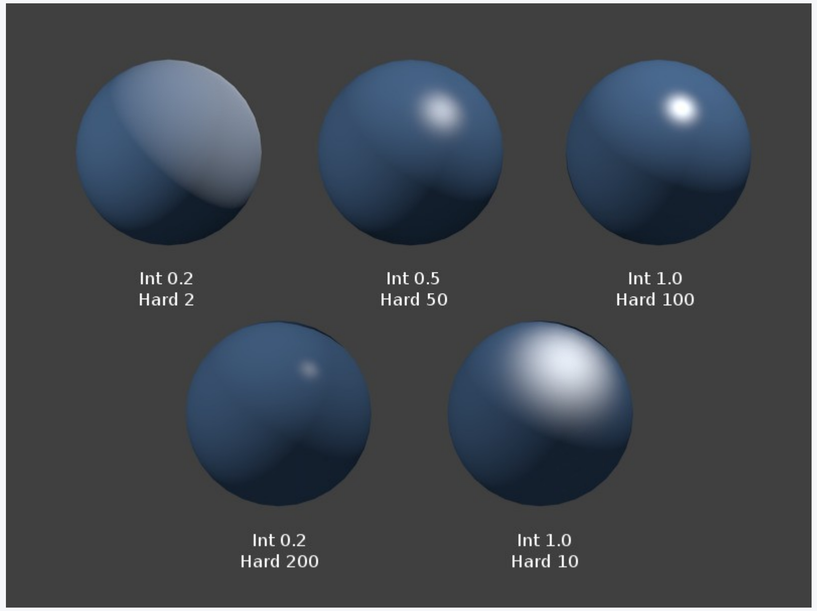
O Qt3D – Materiais

Modelo final combinado:

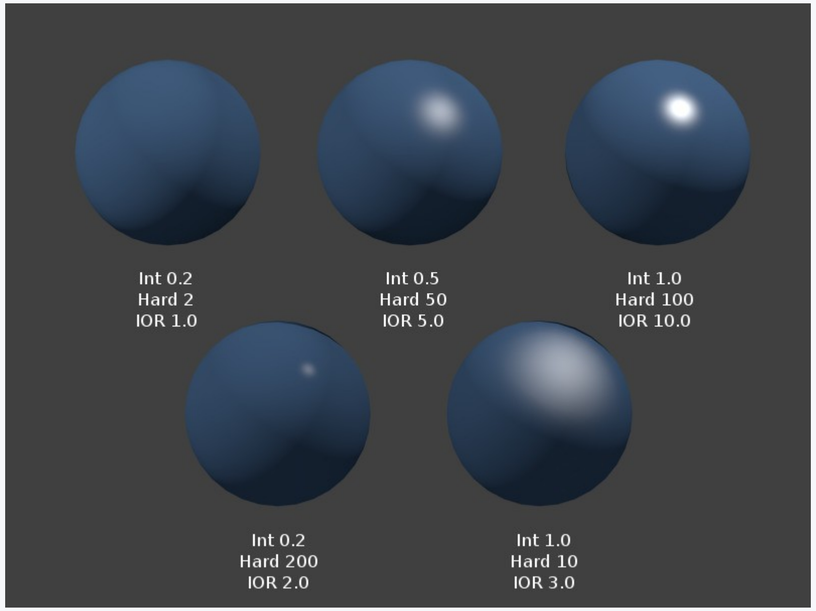
$$I_\mathrm{p} = k_\mathrm{a} i_\mathrm{a} + \sum_{m \in \text{lights}} \left(k_\mathrm{d} (\hat{L}_m \cdot \hat{N}) i_{m,\mathrm{d}} + k_\mathrm{s} (\hat{R}_m \cdot \hat{V})^\alpha i_{m,\mathrm{s}} \right).$$

O Qt3D – Materiais

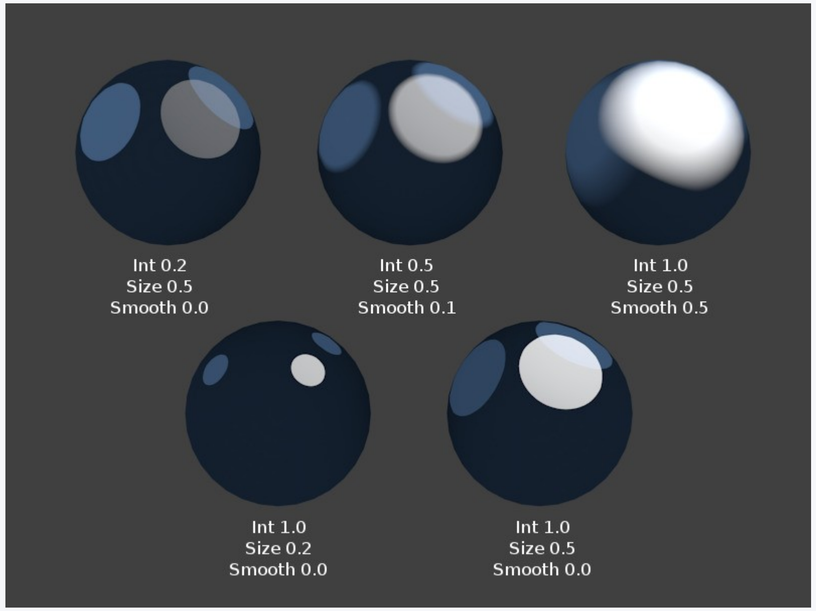
Outros *shaders* especulares:



COOK-TORRANCE



BLINN



TOON

O Qt3D – Materiais

O tipo QML **PhongMaterial**:

– Implementa o modelo de *shading* de Phong.

– Disponibiliza quatro parâmetros: **ambient** (color), **diffuse** (color), **specular** (color) e **shininess** (real).

– A ponderação é realizada através dos valores RGB de cada parâmetro do tipo color.

– O Qt implementa o Phong *shading* com uma única passada de renderização e disponibiliza *shaders* de fragmento para OpenGL 2, OpenGL3+ e OpenGL ES.

O Qt3D – Mapeamentos

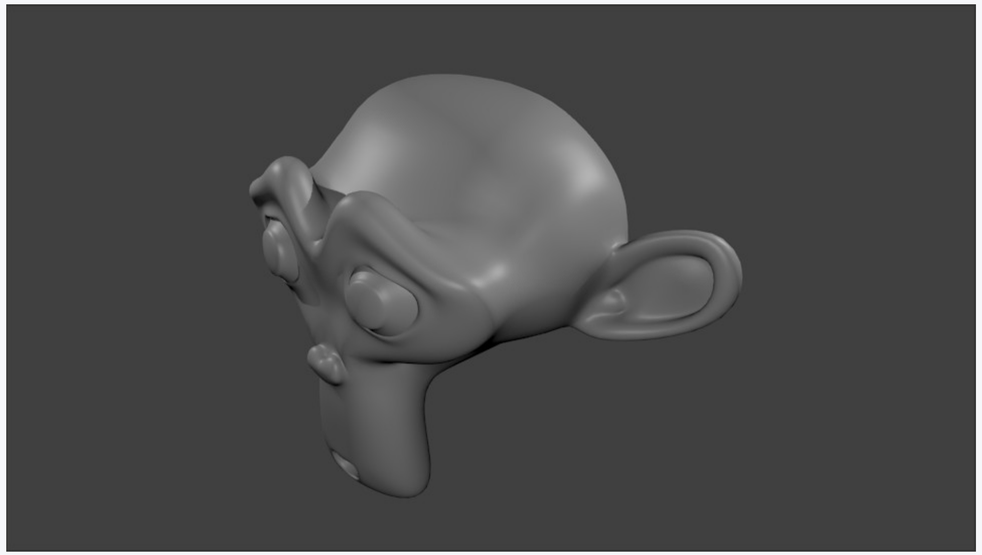
Diversos tipos de mapeamentos complementam a definição de um material:

- **Texture Mapping**
- **Height Mapping**
- **Bump Mapping**
- **Normal Mapping**
- **Displacement Mapping**
- **Specular Mapping**
- **Reflection (Environment) Mapping**
- **Occlusion Mapping**

O Qt3D – Mapeamentos

Diversos tipos de mapeamentos complementam a definição de um material:

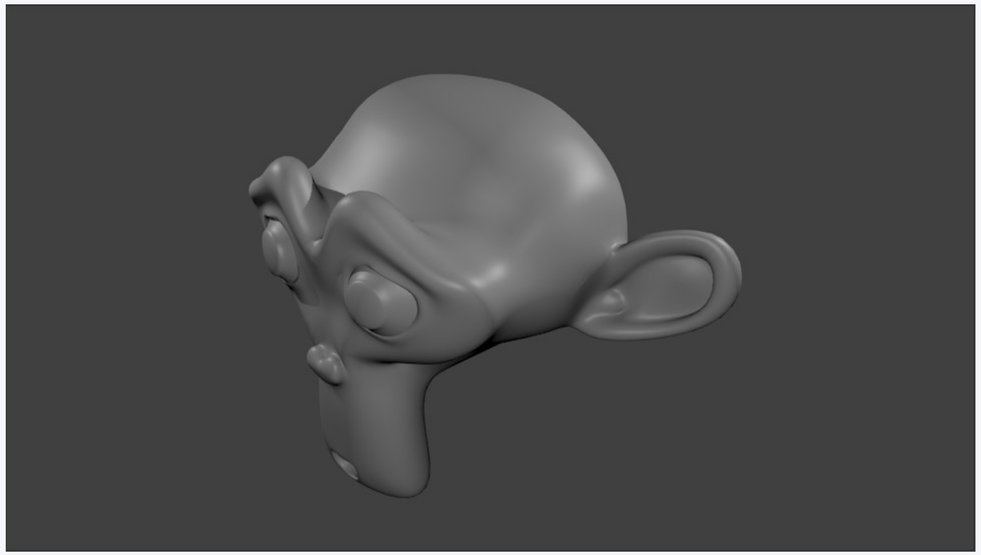
- **Mapeamento de Textura:** método para definir detalhes de alta frequência, textura de superfície ou informação de cor em modelos 3D.

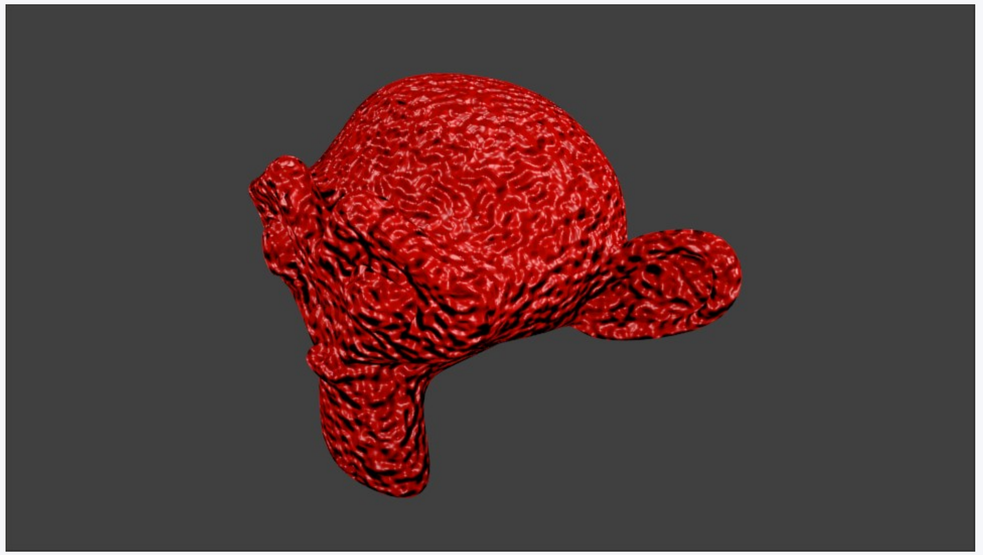


O Qt3D – Mapeamentos

Diversos tipos de mapeamentos complementam a definição de um material:

- **Bump Mapping:** método para simular rugosidade e aspecto áspero em superfícies 3D.





Texture Mapping

```
...
Entity {
    components: [
        PlaneMesh { width: 20; height: 20 },
        DiffuseMapMaterial {
            diffuse: TextureLoader {
                source: "wood.jpg"
            }
            textureScale: 2
        }
    ]
}
```

```
...
Entity {
    components: [
        Mesh { source: "suzanne.obj" },
        DiffuseMapMaterial {
            diffuse: TextureLoader {
                source: "fur.jpg"
            }
        }
    ]
}
```

OBS:

Certifique-se que a sua malha possui um mapa UV associado antes de exportá-la como arquivo .obj.

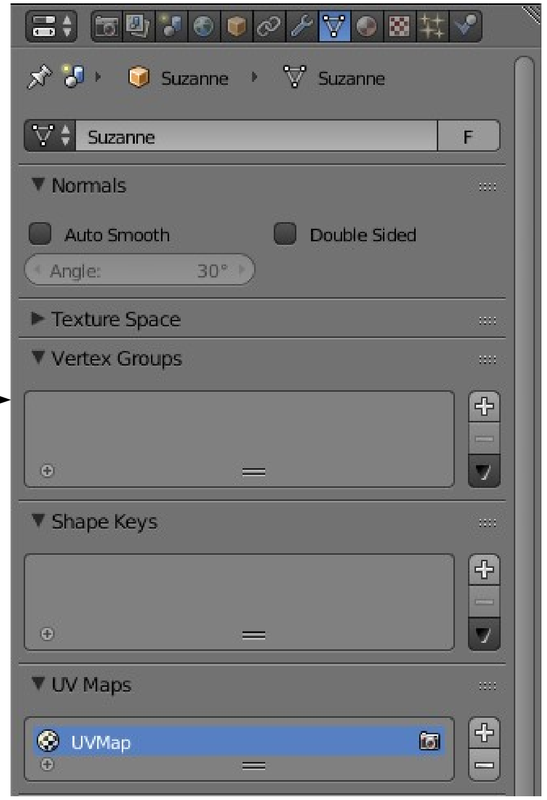
Texture Mapping

No Blender:

- Entre no edit mode (TAB)
- À esquerda, selecione “Shading / UV”
- Em “UV Mapping”, troque de “Unwrap” para “Cube Projection”
- Confirme o mapa UV no painel de dados do objeto

OBS:

Certifique-se que a sua malha possui um mapa UV associado antes de exportá-la como arquivo .obj.



Bump Mapping

```
...
Entity {
    components: [
        Mesh { source: "suzanne.obj" },
        NormalDiffuseMapMaterial {
            diffuse: TextureLoader { source: "fur.jpg" }
            normal: TextureLoader { source: "normal.jpg" }
        }
    ]
}
```

Environment Mapping (I)

```
1. ...
2. SkyboxEntity {
3.     baseName: "qrc:/miramar"
4.     extension: ".webp"
5. }
```

CUBE MAPPING

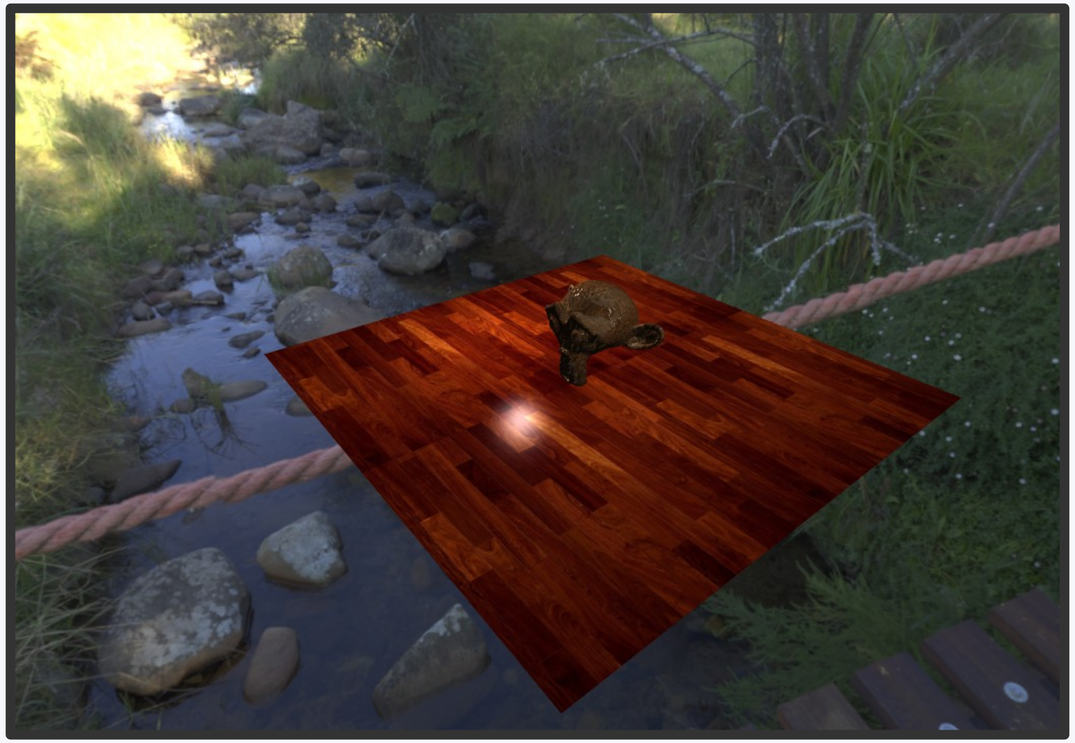
OBS:

Certifique-se que o *frustum culling* esteja desabilitado no framegraph e que a projeção de câmera seja perspectiva.

QtCon BR

Environment Mapping (II)

```
1.  ...
2.  Camera {
3.      id: camera
4.      position: Qt.vector3d(0, 0, -40)
5.      viewCenter: Qt.vector3d(0, 0, 0)
6.      exposure: 1.7
7.      fieldOfView: 60
8.  }
9.  ...
10. SkyboxEntity {
11.     baseName: "wobbly_bridge_4k_cube_radiance"
12.     extension: ".dds"
13.     gammaCorrect: true
14. }
```



O Qt3D – Physically Based Rendering

O Qt 5.9 introduziu novos materiais para PBR com foco em fotorrealismo:

- **MetalRoughMaterial:** PBR + shading especular de Blinn-Phong.
- **TexturedMetalRoughMaterial:** PBR + shading especular de Blinn-Phong + textura difusa.

DEMO (pbr)

NodeInstantiator

DEMO (nodeinstantiator)

Animações com KeyFrame (I)

DEMO (baked-keyframe-animation)

Animações com KeyFrame (II)

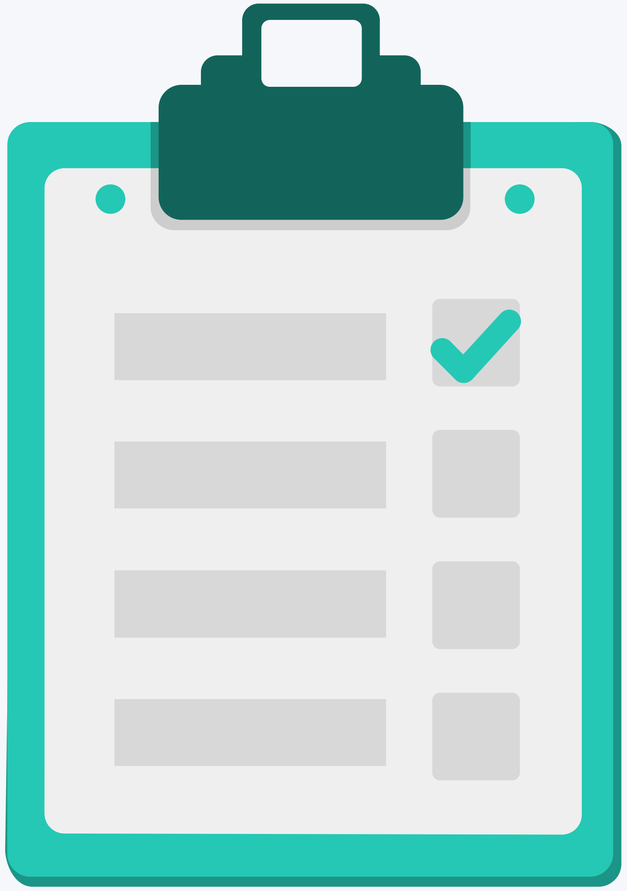
DEMOS

(animated-skinned-mesh e skinned-mesh)

Integração com o QtQuick

DEMO (scene2d e pbr-sphere)

Conclusão



1. O Qt3D encontra-se já bastante maduro para uso em produção.
2. A implementação baseada em OpenGL proporcionada pelos diversos backends faz do Qt3D uma solução interessante em diversas plataformas.
3. A cada nova versão do Qt novidades são acrescentadas no Qt3D.
4. Utilizem o Qt3D e discuta sua experiência no nosso canal “Qt Brasil” no Telegram (http://t.me/qtbrasil).

QtCon BR

QtCon BR

Obrigado!

Sandro S. Andrade
sandroandrade@kde.org

IFBA/KDE