Defesa do Projeto de Graduação – 28/02/2011

Engenharia de Computação e Informação – Poli/COPPE – UFRJ

Orientador:

Prof. Ricardo Marroquim, D. Sc.

Examinadores:

Prof. Claudio Esperança, Ph. D.

Prof. Antonio de Oliveira, D. Sc.

Experimentação em Simulações Físicas Interativas para Jogos

João Pedro Schara Francese

Roteiro

- Motivação
- Objetivo
- Metodologia
- Bibliotecas
  - Panda3D
  - Open Dynamics Engine
- Modelagem Física
- Implementação
- Demo
- Conclusões e Trabalhos Futuros

Motivação

- Popularidade dos jogos eletrônicos
  - US$ 57 bi *vs* 30 bi (cinema) e 17 bi (música)
- Projetos complexos, concorrência forte
- Como conquistar o consumidor: imersão
  - História
  - Realismo nos gráficos
  - **Interação realista (física)**

Crysis (2007)

Objetivo

- Implementar uma *demo* de um jogo de tiro em primeira pessoa (FPS)
  - Reações físicas realistas
  - Manter interatividade e tempo real
  - Capaz de ser executado em hardware atual
- Estudo das ferramentas e técnicas utilizadas

Metodologia

1. Seleção das tecnologias a serem usadas
   - Software livre, adoção comprovada, documentação online
2. Definição da funcionalidade-alvo
   - Destruição dinâmica de paredes do cenário
3. Definição de critérios de medição e metas
   - Realismo físico (subjetivo)
   - Quadros por segundo: 30QPS (repouso) e 15QPS (destruição de um tijolo)
4. Desenvolvimento

Motores de Jogos

- Componentes que abstraem tarefas rotineiras ou de baixo nível e ocultam diferenças entre as plataformas
- SDK (*software development kit*): interface + biblioteca + ferramentas
- Normalmente atrelada a uma linguagem
- Restrições: multiplataforma e código aberto

Panda3D

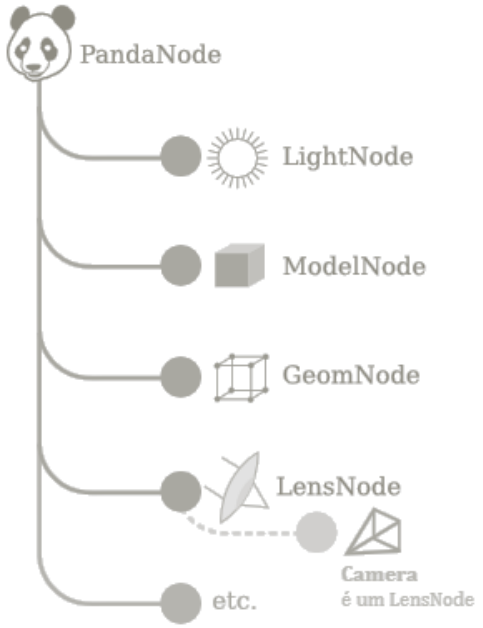
- Motor para jogos tridimensionais
- Foco na gerência dos gráficos
  - Permite tanto um uso simplificado quanto técnicas avançadas (ex.: *shaders*)
- Diversas funcionalidades auxiliares
  - Temporização, áudio, rede, física, I.A.
- Curso de Animação e Jogos da UFRJ
- http://www.panda3d.org/

Panda3D

- Linguagem: **Python**
  - Multiplataforma, desenvolvimento ágil
  - Código interno escrito em C++
    - Custo para fazer troca de contexto
- Licença BSD
- Desenvolvido por Disney e universidade Carnegie Mellon
  - Uso em jogos comerciais de realidade virtual da Disney

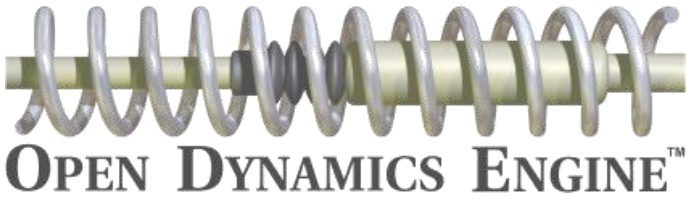
Panda3D

- Estruturação em um **grafo de cena**
  - Cada modelo é um nó, assim como luzes, a câmera e outros objetos
  - Transformações e propriedades são herdadas e propagadas de pai para filho
    - Código enxuto
    - Composição de objetos complexos
  - Árvores com duas raízes: *render* e *render2d*



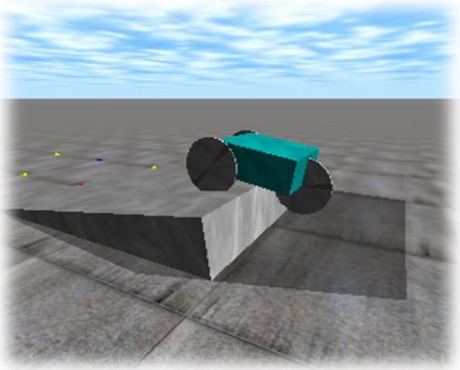
Open Dynamics Engine

- Motor de simulação física + detector de colisões entre objetos
- Independente do motor gráfico
  - Também é possível terceirizar tratamento de colisões
- Simulação interativa
  - Não garante tempo real
- http://ode.org/



Open Dynamics Engine

- Modela ambientes de realidade virtual
  - Veículos, objetos móveis e criaturas articuladas
- Linguagem: **C++**
  - PyODE e Panda3D
    - Versão antiga do ODE
- Licenças BSD e GPL
- Além de atender aos requisitos, possui integração com o Panda3D
- Usado em Call of Juarez e World of Goo

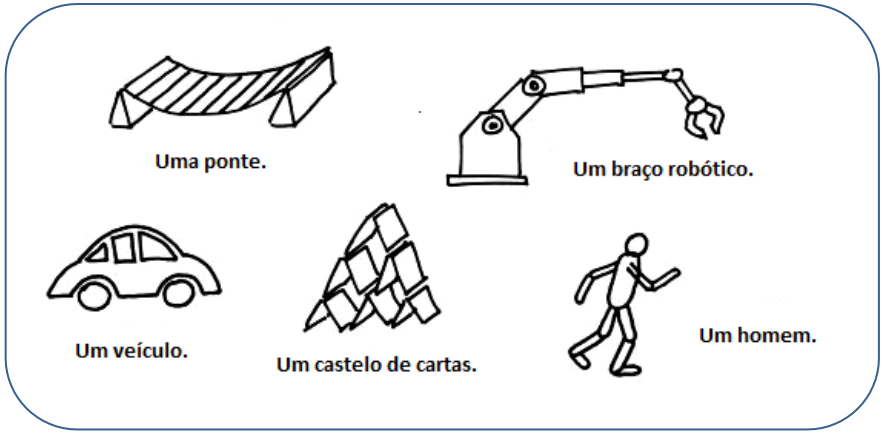


Modelagem Física

- Poder computacional é finito → aproximações e simplificações (obrigatoriamente)
- Simulação em cinco passos:
  1. Entender o sistema
  2. Modelar através de equações
  3. Escrever um algoritmo
  4. Implementar em um programa
  5. Executar a simulação

Modelagem Física

- Dinâmica de corpos rígidos articulados
  - *Dinâmica:* interação e movimentos
  - *Corpos rígidos:* objetos sólidos que não sofrem deformação
  - *Articulados:* uso de junções para unir corpos e impor restrições

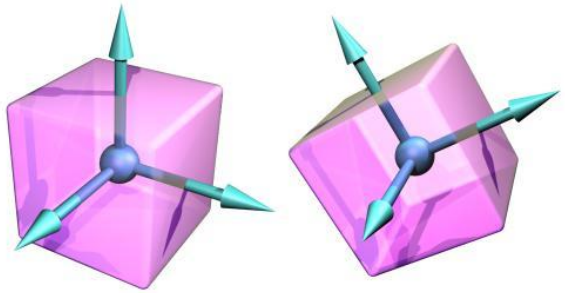


Modelagem Física

- Estrutura de código mínima com ODE
  - Criar um mundo dinâmico
  - Criar corpos e definir seu estado inicial
  - Criar junções e ligar aos corpos
  - Criar mundo de colisão e geometrias de colisão
  - Executar em laço, até o fim do jogo:
    - Aplicar forças
    - Ajustar parâmetros das junções
    - Chamar rotinas de detecção de colisão*
    - Criar junções de contato nos pontos de colisão*
    - Rodar um passo da simulação física
    - Remover junções de colisão
- Alguns itens tratados pelo Panda3D

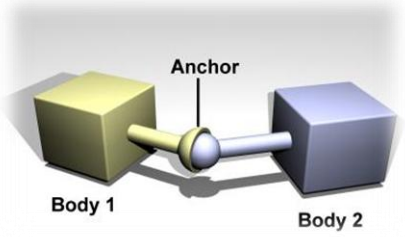
Modelagem Física

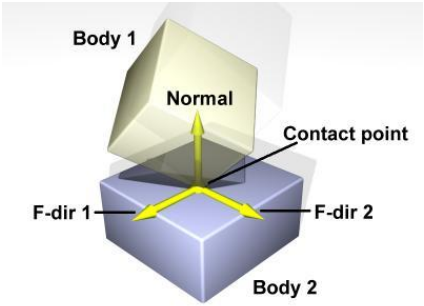
- Corpos rígidos
  - Sem deformação = distância constante entre dois pontos do corpo
  - Propriedades estáticas (massa, posição relativa do centro de massa, matriz de inércia)
  - Propriedades dinâmicas (vetores de posição, velocidade, orientação e velocidade angular)
  - Eixo de coordenadas

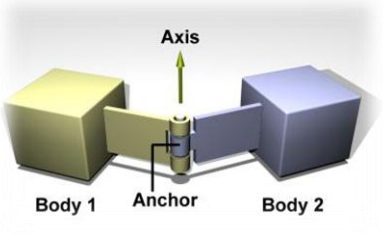


Modelagem Física

- Junções
  - Restrições nas propriedades relativas entre corpos
  - Passo de simulação gera forças sobre corpos para satisfazer as junções
  - Vários tipos: fixa, bola e encaixe, dobradiça, contato, motores...







Modelagem Física

- Forças e torque
  - Vetor de 3 elementos
  - Acumulador recebe forças inseridas pelo programador a cada passo
  - Integrador calcula novas velocidades e posições
    - ODE: estável (para Δt constantes) e preciso (para Δt pequenos)
  - Programador pode obter força resultante
    - Mas não no Panda3D

Modelagem Física

- Tratamento de colisões
  1. **Detecção**
     - Métodos fornecidos pelo ODE
     - Agrupamento em categorias (*bitmasks*)
     - Não testa pares distantes entre si
  2. **Reação**
     - Programador cria junções de contato*
     - Usuário define parâmetros para a colisão*
     - ODE insere restrições no integrador
     - Outras bibliotecas: molas

Modelagem Física

- Parâmetros de colisão
  - *mu:* coeficiente de fricção
  - *bounce:* coeficiente de restituição elástica
  - *bounce_vel:* velocidade mínima para rebater
  - *slip:* permite que corpos deslizem entre si
  - *dampen:* amortecimento em oscilações
  - *soft_erp/soft_cfm:* correção de erro
- Geometrias de colisão
  - Caixa, cilindro, cápsula, plano, raio, esfera, *trimesh*
  - Propriedades geométricas, mas não dinâmicas

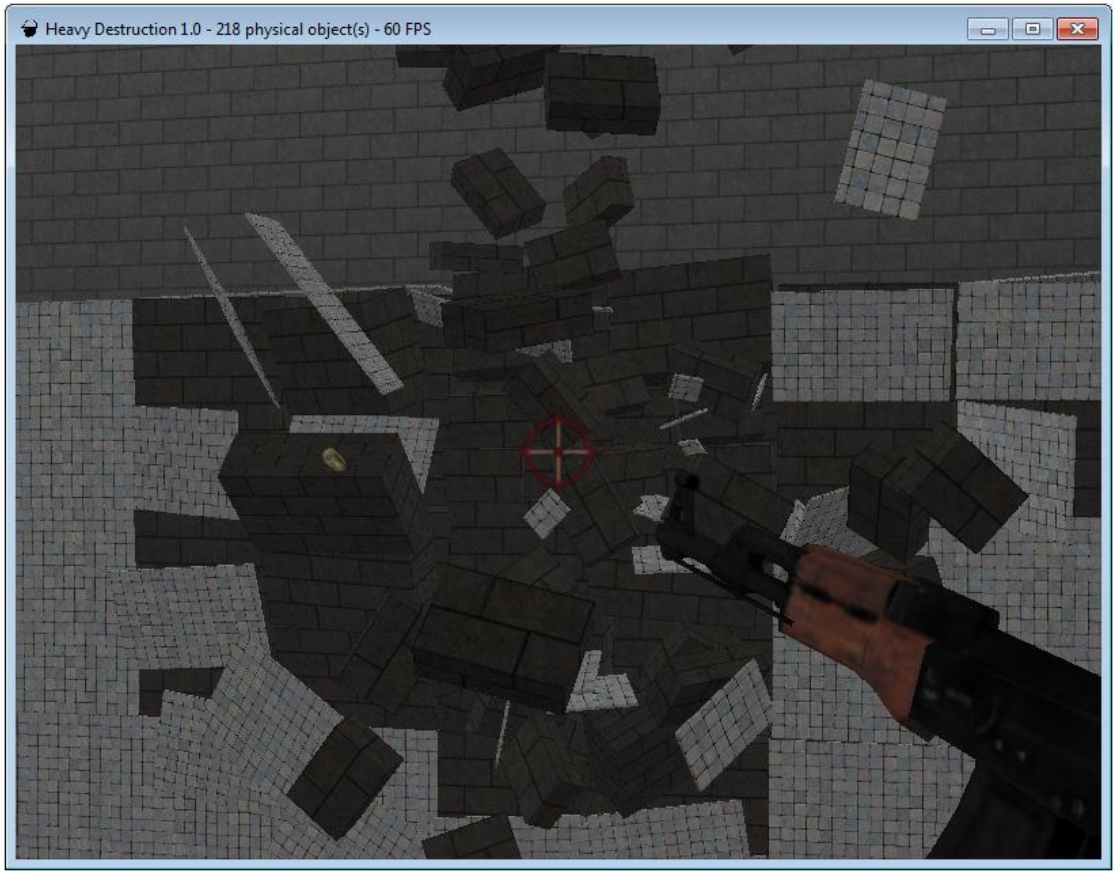
Implementação

- Demo *vs* jogo
- Cenário com limites indestrutíveis e uma parede especial
  - Tijolos e azulejos que podem ser derrubados ou quebrados
- Controles: andar (WASD), pular (espaço), atirar (mouse)
- Pausa na simulação com ESC
- Cena alternativa: "*falling balls*"

Implementação

- Linguagem Python, IDE NetBeans
- Orientação a objetos
- Testes em laptop médio-avançado atual
  - Windows 7 x64, Intel Core i5 2.53GHz, memória de 6GB, GeForce GT330M com 1GB
- Mídia de uso livre: http://opengameart.org/

Demo



http://code.google.com/p/heavy-destruction/
http://tiny.cc/destruction

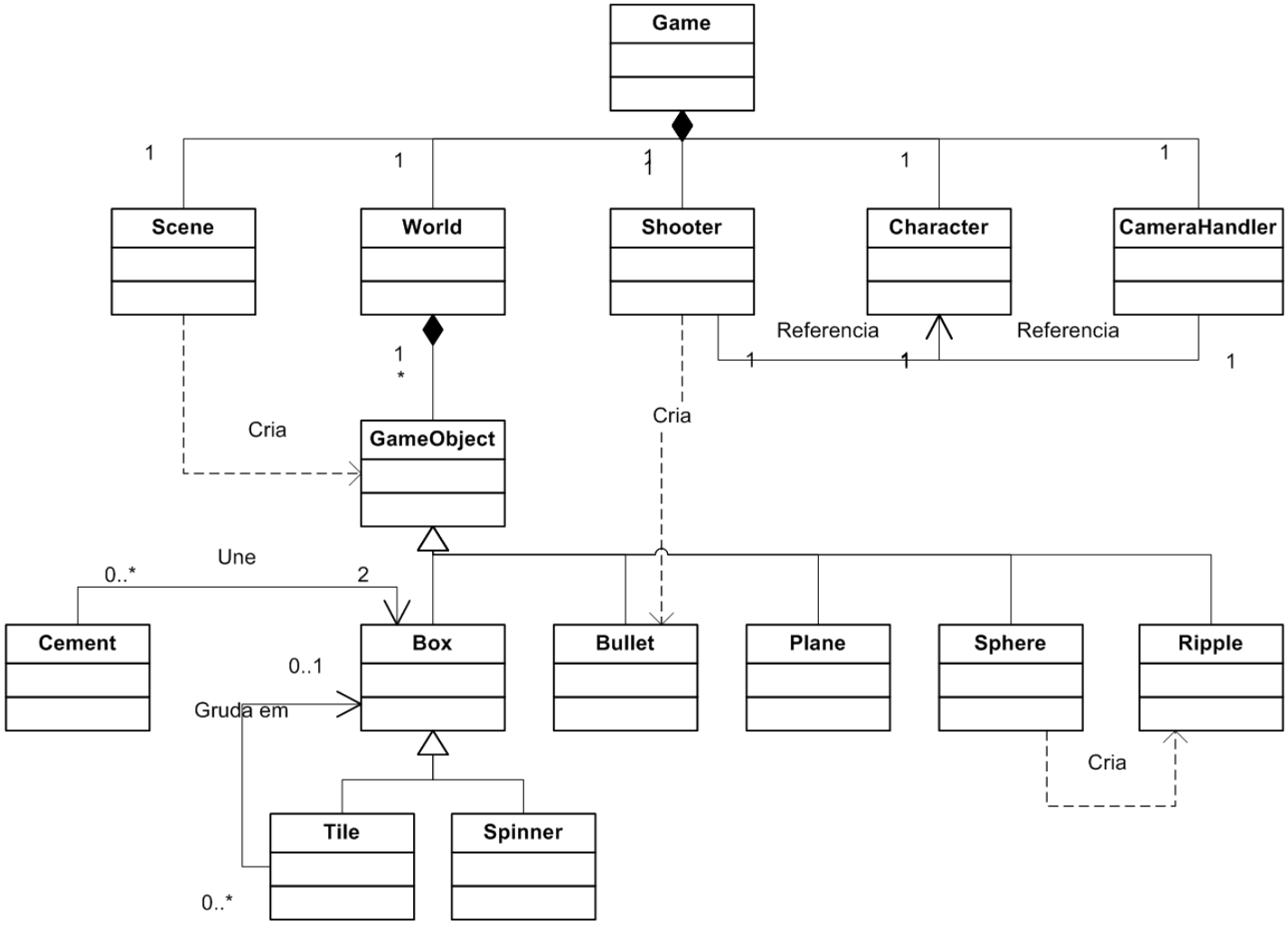
Implementação

- Classe *World*
  - Container dos objetos do jogo
  - Responsável pela simulação física
- Classe *GameObject*
  - Representa um objeto físico
  - Construtor, destrutor e auxiliares (ex.: *GetMomentum*)
- Classe Scene
  - Povoa o mundo com objetos

Implementação

- Classe Character
  - Controla o movimento do personagem e o salto
  - Objeto físico
- Classe CameraHandler
  - Controla a câmera e movimentação do mouse
  - Mesma posição do jogador
- Classe *Shooter*
  - Disparo da bala na posição do jogador e direção da câmera
  - Modelo da arma

Implementação

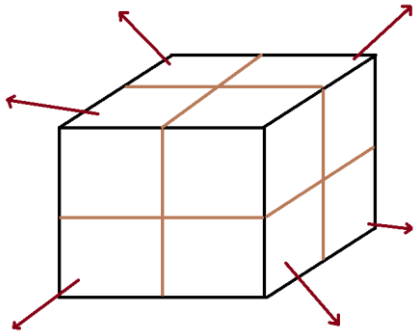


Implementação

- Classes *Box* e *Tile*
  - Modelo com textura
  - Paralelepípedo (*geom*)
  - Tijolos colados entre si com a classe *Cement*
    - Junção do tipo Fixa
    - Remoção automática
  - Azulejos inicialmente colados a um tijolo

Implementação

- Quebra de tijolos e azulejos:
  - Condição:velocidade linear > limiar
  - Efeito: troca por objetos menores
  - Velocidade para fora + velocidade dos objetos que colidiram
  - Limite definido no construtor
  - Azulejo pode descolar sem rachar



Implementação

- *Bullet*
  - Cilindro
  - Sem gravidade inicial
  - Dissipação artificial de energia ao colidir
  - Inicialmente invisível
  - Velocidade de 20m/s a 60m/s, angular de 80º/ a 400º/s
- Outros objetos:
  - *Plane*
  - *Spinner*, *Sphere*, *Ripple*

Implementação

- Imersão além da física:
  - Música ambiente
  - Efeitos sonoros ao caminhar e saltar
  - Lampejo e efeito sonoro ao atirar
  - Recuo ao disparar bala
    - Jogador: físico (velocidade para trás)
    - Arma: programado
  - Vibração na tela e efeito sonoro ao quebrar tijolo

Otimizações

- Remoção de objetos inativos
  - Objetos inseridos ao longo do jogo → queda na taxa de QPS
  - Remoção de balas após número de colisões com o chão
  - Remoção de tijolos e azulejos advindos de quebra
    - Em repouso (velocidade ~= 0)
    - Sem colisão

Otimizações

- Colisão condicional
  - Detecção de colisões é $O(n^2)$
  - Após introduzir azulejos, taxa em repouso ficou inferior a 30QPS
  - Solução: criar categorias de objetos
    - Azulejos colados não colidem com nada exceto balas
    - Jogador não colide com balas
- Limitação do tempo gasto com passos de simulação física

Conclusões

- Simulação: compromisso entre qualidade e velocidade
  - Qualidade: quantidade de objetos + realismo
- Testar cada otimização
  - Nem toda tentativa resulta em melhora
- Reações positivas a exibições confirmaram os efeitos de uma simulação física realista

Trabalhos Futuros

- Ampliação da demo
  - Simulação apenas para objetos próximos
  - Agrupamento espacial de objetos
- Outras simulações
  - Corpos deformáveis, tecidos, fluidos, partículas
- Implementação com outras bibliotecas
  - Bullet Physics e NVIDIA PhysX

Defesa do Projeto de Graduação – 28/02/2011

Engenharia de Computação e Informação – Poli/COPPE – UFRJ

Orientador:

Prof. Ricardo Marroquim, D. Sc.

Examinadores:

Prof. Claudio Esperança, Ph. D.

Prof. Antonio de Oliveira, D. Sc.

Experimentação em Simulações Físicas Interativas para Jogos

João Pedro Schara Francese