

Experiência de um Curso Prático de Introdução à Programação em Linguagem C++ Aplicada ao Desenvolvimento de Jogos em 3D

MARCELO DE OLIVEIRA JOHANN
MARCUS KINDEL
CINTYA BORTOLAZZO

Pontifícia Universidade Católica do Rio Grande do Sul - Campus Universitário II
Faculdade de Administração, Contabilidade e Informática - Departamento de Informática
{johann, kindel, cintya}@pucrs.campus2.br

Abstract

This paper describes an introductory course on object-oriented programming applied to the development of computer games with 3D modeling. The use of games as the goal allows us to approach both basic concepts and complex practical problems of programming, while keeping the students interested on the subject. A survey of standard techniques for game development are presented with running examples at the course, although the objective is not to cover all aspects of computer graphics, artwork, and other advanced techniques required for finished products. The paper describes the motivation, objectives, the structure and several aspects of the development and application of this 64 hours undergraduate course, as well as the resources used to implement it.

Keywords: *programming, object orientation, C++, education, games*

1 Introdução

Jogos de computador apresentam uma necessidade acentuada de poder computacional. A utilidade desses jogos pode ser questionável, mas do ponto de vista tecnológico, programar um jogo consiste em um desafio comparável aos mais complexos da ciência da computação. Os recursos visuais, sonoros, e de controle desejados estão sempre além da tecnologia disponível. Por esta razão, os programadores de jogos sempre extraíram o máximo da tecnologia existente, por exemplo, usando modos de operação especiais dos processadores de vídeo, programando em linguagem *assembly* para ter código mais otimizado, e principalmente usando tabelas de resultados pré-computados. Algumas dessas técnicas são dificilmente dominadas por programadores iniciantes ou mesmo por aqueles já experientes em outras áreas.

Os primeiros jogos disponíveis utilizavam apenas modelos bidimensionais ou quase unidimensionais, onde o protagonista pode apenas mover-se para a direita ou esquerda. Com a evolução tecnológica, muitos jogos atualmente empregam o que há de mais moderno em *hardware* e *software* para apresentar universos virtuais em 3 dimensões (3D) com uma infinidade de recursos. Os processadores gráficos de placas de vídeo para computadores domésticos possuem hoje a capacidade de desenho de milhões de faces em 3D (triângulos) por segundo, usando efeitos de textura, iluminação, atmosfera, fazendo recorte, transformações, cálculo de visibilidade, transparência, etc. automaticamente. Esse fator já simplifica muito a programação de jogos.

Além de plataformas de *hardware* mais poderosas, também surgiram as plataformas de *software* para o desenvolvimento de jogos, denominadas de motores ou *engines*^{3,4}. Os motores são bibliotecas de *software* que implementam conceitos e facilidades necessárias à maioria dos jogos, como representação de um ambiente, montagem do quadro a desenhar, controle de eventos e de tempo, além de uma série de funções genéricas de programação. Com essas plataformas, torna-se possível desenvolver aplicações visualmente ricas em níveis de abstração bem mais altos.

A contínua evolução de plataformas não irá eliminar completamente a necessidade de economizar recursos, levada ao extremo nos primeiros jogos, mas acredita-se que os profissionais do futuro devam estar preparados para trabalhar com modelos de jogos complexos em 3D, e por isso deve ser dada ênfase ao ensino dessas tecnologias.

Do ponto de vista de *software*, um jogo é bastante semelhante a processos de simulação e sistemas de tempo real. Um processo de simulação deve acompanhar a geração e execução de eventos no tempo, testando condições segundo as regras do domínio em questão. Um sistema de tempo real tem como principal característica o fato de que deve atender a determinadas restrições de temporização externa.

Do ponto de vista de projeto, as tecnologias mais importantes de programação são hoje intensivamente utilizadas na implementação de jogos. São praticamente indispensáveis: orientação a objetos, *design patterns*², *templates*¹, *containers*¹ e iteradores¹, ponteiros seguros, tratadores de eventos, sistemas de janelas, programação concorrente, comunicação, bases de dados, etc. Isto quer dizer que a maior parte das tecnologias genéricas avançadas de programação e desenvolvimento de *software* são

empregadas no projeto de um jogo, e não mais apenas um trabalho artesanal em *assembly*, enquanto que, por outro lado, o talento de dominá-las da maneira mais eficiente e diversificada continua presente. São esses fatores que tornam tão atraente estudar a programação de jogos do ponto de vista computacional.

Além do uso de *hardware* e domínio da tecnologia de *software*, há o projeto gráfico e o projeto semântico do jogo, que tendem a requerer a maior parte do esforço de projeto, principalmente em jogos com modelos tridimensionais. Em nosso caso, todo esse trabalho de arte e roteiro não é de interesse particular.

2 Motivação

Conceitualmente, um bacharel em Ciência da Computação é um profissional muito superior a um simples programador especializado em codificar pequenos algoritmos em uma determinada linguagem. Entretanto, a maioria do conteúdo computacional é expresso sob forma de programas, mesmo em casos onde este seja posteriormente implementado em *hardware*. Assim, é de fundamental importância que um bacharel tenha destreza em programação utilizando as mais variadas técnicas e linguagens disponíveis.

Apesar desta necessidade básica, o aprendizado de programação não é trivial, leva muito tempo para ser desenvolvido, e muitas vezes não atinge um grau de maturidade mesmo para alunos que se formam em nossas Universidades. É conhecida a dificuldade de alunos principiantes para compreender técnicas básicas de algoritmos e programação, e as necessidades e vantagens de técnicas mais avançadas como orientação a objetos ou outras abstrações se eles não tiveram uma boa experiência. Por esta razão, os professores universitários estão continuamente em busca de novas técnicas de ensino, e principalmente de novos problemas que motivem os alunos, partindo da aplicação, da necessidade, para que eles entendam a utilidade de tão complexos mecanismos, como os disponíveis em linguagens como C++¹.

A aplicação de jogos tem esta característica especial de atração para os jovens estudantes. A maioria dos estudantes já jogou ou é acostumada a jogar regularmente jogos de computador e *video games*. E como todos os jovens, os alunos têm curiosidade e criatividade potenciais para trabalharem em assuntos relacionados aos seus maiores interesses, ou àquilo em que vêem atração, não fazendo parte de suas obrigações.

Segundo pesquisa realizada no Campus de Uruguaiana, a grande maioria dos alunos demonstrou forte interesse em aprender técnicas usadas para a construção de jogos. Tendo em vista que estas técnicas incluem, obrigatoriamente, a maior parte do conteúdo de programação que se deseja ensinar, o potencial de aprendizado por ela oferecido é muito animador, e espera-se que seu emprego venha a sanar deficiências inerentes a outras abordagens.

3 Objetivos

No contexto exposto acima, o presente trabalho descreve um curso de extensão universitária em nível de graduação, oferecido a alunos que já cursaram disciplinas básicas de algoritmos, estruturas de dados, programação e que têm noções básicas de

orientação a objetos, e desejam conhecer, reforçar, retirar dúvidas, dominar a programação orientada a objetos em linguagem C++ e tomar contato com as técnicas de desenvolvimento de jogos de computador com modelagem em 3 dimensões. Assim, destacam-se dois objetivos gerais fundamentais, e os objetivos específicos definidos abaixo.

Objetivos Gerais:

- Ensinar e praticar a programação orientada a objetos em C++;
- Ensinar as tecnologias para desenvolvimento de jogos com modelos 3D.

Objetivos Específicos:

- Desenvolver capacidade de definir classes e programar usando objetos;
- Apresentar motores 3D disponíveis e como se os utiliza;
- Conhecer a arquitetura e temporização de um jogo;
- Conhecer e saber como usar as técnicas de modelagem e visualização em 3D, movimentação, tratamento de eventos, animação, detecção de colisões, física, representação de terrenos e superfícies;
- Dar uma visão da indústria e do mercado de jogos, objetivos e fatores de sucesso.

4 Metodologia de Ensino

O curso oferecido tem 64 horas-aula práticas, em laboratório, com computadores individuais, e é todo baseado em exemplos. Os exemplos iniciais são bastante simples, com o menor número de linhas de código que permita criar e exibir objetos tridimensionais. Cada exemplo é experimentado e alterado de diversas formas pelos alunos para dar aos mesmos o domínio do programa, de seu funcionamento, e do que está expresso na linguagem de programação.

Os desafios lançados aos alunos, como, por exemplo, completar um determinado objeto que está pela metade, estimulam sua criatividade e desejo de programar cenas cada vez mais complexas, assim permitindo-lhes dominar as técnicas ensinadas e, ao mesmo tempo tirarem suas próprias conclusões enquanto aprimoram suas técnicas de programação. A partir destes exemplos, os demais conceitos são inseridos um a um, construindo ambientes mais complexos, até incluírem movimentos, controle, e se assemelharem a jogos simples. O processo é repetido várias vezes com novos objetivos.

Após estas experiências, são propostos problemas mais genéricos de jogos, como movimento simultâneo de diversos objetos, e então apresentadas algumas opções de arquitetura do programa para tratar esta simulação em tempo real. Dá-se prioridade ao estudo de uma arquitetura que atenda às necessidades, para que também com ela os alunos se acostumem, embora saibam que há outras formas possíveis de estruturar o programa. Durante todo este processo, a orientação a objetos é usada intensivamente, já que os ambientes com que se trabalha são 100% baseados nela. Por esta razão os exemplos simples iniciais são conduzidos, passo a passo, para firmar o domínio básico de orientação a objetos. Assuntos mais avançados ou exceções devem ser evitados tanto

quanto possível. O objetivo principal é dar poder seguro de expressão com orientação a objetos, tornando-a uma maneira intuitiva de pensar.

Finalmente, são abordados os assuntos mais avançados da tecnologia de jogos, como uso de animações, detecção de colisões e sistemas de física, descrições de superfície e terrenos e uso de diferentes níveis de detalhe, também com uso de exemplos práticos, embora não haja tempo para construir um jogo completo que envolva todos estes recursos.

5 Infra-estrutura

Para realização do curso, é utilizado um laboratório com 20 máquinas (Pentium III 1.3GHz e Pentium IV 1.7GHz), todas com 128 MB RAM, placa aceleradora gráfica e sistema operacional Linux Red Hat 7.3.

Os *engines* utilizados são o **Gizmo3D**³ e **Crystal Space**⁴. As licenças para uso desses sistemas são: livre para Crystal Space (LGPL) e comercial ou livre (para fins não comerciais) para o Gizmo3D. Tanto um como outro sistema oferecem quase a totalidade das características descritas a seguir:

Plataformas Linux e Win32, tratamento de exceções, algumas *design patterns*, *templates* para dicionários, listas, filas, vetores dinâmicos; clonagem, *threads* e *mutexes*, *RTTI*, *rendering* de múltiplas passagens, incluindo iluminação, texturas, sombras, atmosfera, transparência; superfícies paramétricas com níveis de detalhe automáticos, seleção e detecção de colisões, controle de animações, e sistemas de bases de dados para leitura de alguns formatos de imagens, animações e mapas.

Na primeira metade do curso os alunos trabalham com o Gizmo3D, pois este possibilita a visualização dos resultados com apenas poucas linhas de código. Isso favorece aos alunos a compreensão de cada mecanismo apresentado. Na segunda metade do curso, já com o domínio de algumas técnicas fundamentais, os alunos poderão trabalhar com o Crystal Space, *engine* que oferece construções mais poderosas. Este não é utilizado primeiramente porque a estrutura dos programas é bem mais complexa, e os alunos demoram para dominar as abstrações nele utilizadas.

6 Conteúdo Programático

Os tópicos abordados no curso são, basicamente, os descritos abaixo. A critério dos professores, foi especificado que os itens poderiam ser simplificados, estudados com mais profundidade ou novos tópicos poderiam ser inseridos, dependendo do progresso dos alunos no decorrer do curso. Apesar disso, o planejamento pode ser seguido à risca nos primeiros três meses, variando apenas no último. Seguem os tópicos previstos, organizados por semana:

Sem. 1 – Introdução, ambiente, caracterização do motor **Gizmo3D**, triângulos, câmera, texturas

Sem. 2 – Vértices e arestas, transformações, arquivo de descrição de objetos

Sem. 3 – Funções *callback OnIdle* e *OnTick*

Sem. 4 – Controles de teclado e de movimentos

Sem. 5 – Primeiros jogos – **estoque**

Sem. 6 – Movimentos, Tempo e Arquitetura

Sem. 7 – Movimentos, Tempo e Arquitetura

Sem. 8 – Rede, Mensagens, Jogo de duplas

Sem. 9 – Estrutura do **Crystal Space**

Sem. 10 – Arquivo de mapa, animações

Sem. 11 – Representação de terrenos

Sem. 12 – Detecção de colisões

Sem. 13 – Efeitos diversos (som, neblina, representações procedurais, etc.)

Sem. 14 – Sistemas de Física

Sem. 15 – Superfícies curvas; elaboração de um projeto de jogo;

Sem. 16 – Tipos, Objetivos⁵ e Mercado de Jogos; início da implementação de um jogo;

7 Desenvolvimento

Nas primeiras duas semanas do curso são apresentadas as noções introdutórias sobre jogos, linguagem de programação, modelagem e visualização. Os jogos usam modelos geométricos das superfícies dos objetos, descritas como triângulos. Os triângulos são definidos por pontos em 3D e a sua superfície pode ser descrita por um material, o qual terá comportamento sob luz, ou por imagens de texturas. Os alunos testam a criação de vários objetos simples, sua inclusão na cena, onde os observam através de uma câmera, acostumando-se com as coordenadas e com a visão da projeção. A Fig. 1 apresenta dois modelos 3D criados por alunos após a quarta aula, que já eram indicativos do resultado prático que é desenvolvido, o qual se mantém ao longo de todo o curso.

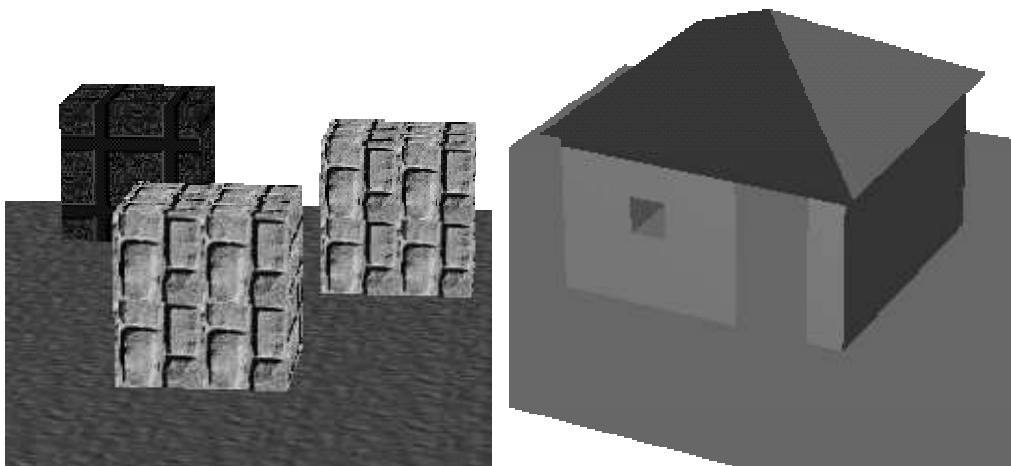


Figura 1 – Modelos simples criados por alunos com textura e iluminação

Quanto à linguagem, os exemplos iniciais são bem organizados, mas usam apenas funções simples para isto, para que sejam os mais parecidos com a linguagem C possível.

As únicas características fundamentais de C++ que são bem apresentadas são a criação de objetos e a chamada dos métodos, exclusivamente pela sintaxe abaixo:

```
Tipo *ponteiro = new Tipo(parâmetros);  
resultado = ponteiro->método(parâmetros);
```

Com essa sintaxe os estudantes podem se acostumar a usar os objetos fundamentais do Gizmo3D: `gzScene`, `gzWindow`, `gzCamera`, `gzNode`, `gzTryGeometry`, `gzVec3`, `gzGroup`, etc. A estrutura que formam é desenhada para que compreendam seu relacionamento.

Na terceira e quarta semanas começa a ser introduzida a definição de classes, de maneira muito suave. A primeira classe introduzida serve somente para que se possa utilizar as funções *callback* do *engine*: *OnTick* e *OnIdle*. É revisto o conceito de interfaces e a necessidade de que o sistema operacional e o sistema de janelas controlem a ferramenta. Logo após, é introduzido o tratamento de eventos de teclado, com o qual eles podem movimentar objetos na cena, usando também as transformações pela primeira vez.

Na quinta semana, para dar sentido ao tema de jogos, é utilizado o exemplo do jogo de **Controle de Estoque**. Nesse jogo, os estudantes vêem a modelagem lógica necessária ao controle, nesse caso representada por um mapa de caracteres. É o primeiro e único exemplo completo fornecido no curso, pois alguns outros que envolvem objetivos finais são parcialmente implementados e sua conclusão sugerida como exercício. A Fig. 2 apresenta uma visão do jogo completo de Controle de Estoque, o qual define novas classes e já usa mais recursos da linguagem C++.

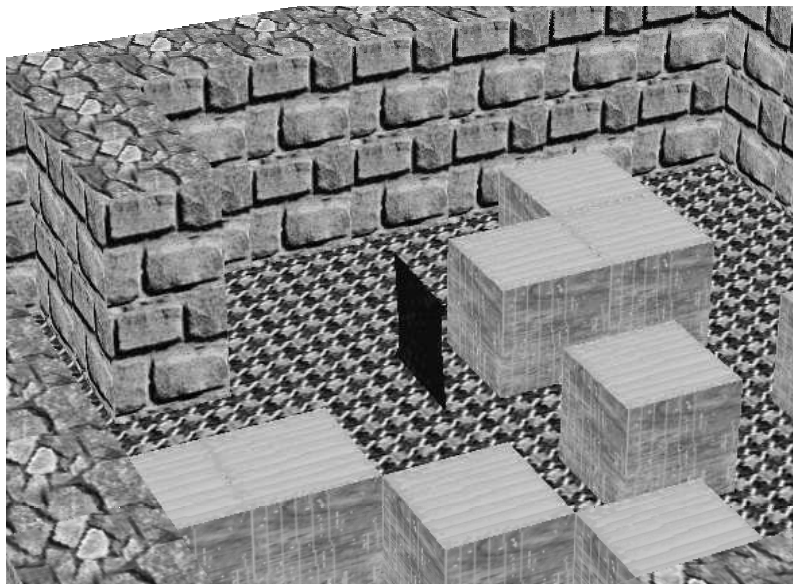


Figura 2 – Jogo “Controle de Estoque”, com funcionalidade completa

Na sexta e sétima semanas, são expostos os problemas de movimento simultâneo e de temporização, para apresentar a arquitetura escolhida do jogo. O problema de movimento é exposto mostrando que dois objetos que devem se mover segundo diferentes controles não podem residir em laços de iteração que representem esses controles, mas sim em um laço genérico, e então que esse não pode existir pela

necessidade de devolver o controle ao sistema de janelas. Assim é introduzido o conceito de que cada objeto deve mover-se um pouco a cada quadro, e que o controle ficará distribuído por essas múltiplas chamadas. É utilizada uma máquina de estados para fazer um automóvel mover-se segundo uma rota retangular, fazendo curvas suaves. Ao final, é introduzida a temporização mostrando que o jogo não pode acelerar-se ou retardar-se de acordo com a complexidade da cena ou com o *hardware* em que roda. Para isso, utilizam-se movimentos calculados com parâmetros reais (velocidade, por exemplo) e controle do *framerate* - quantos quadros o sistema está desenhando por segundo.

A Fig. 3 apresenta uma das últimas versões do exemplo **Drive**, usado para apresentar a arquitetura padrão de um jogo com movimentos simultâneos. Nesse exemplo a câmera move-se desacoplada do carro, seguindo-o, o que confere uma impressão mais suave e dinâmica ao jogo. Além disso, mesmo sem utilizar um sistema de física mais realista, implementou-se o controle do veículo com aceleração constante quando pressionadas as teclas, atuando sobre a velocidade (e esta sobre a posição), e com a direção atuando também indiretamente sobre o sentido real de movimento, além de uma situação de derrapagem onde o móvel vence a força de atrito lateral.



Figura 3 – Exemplos “Drive”, com movimentos e cenário mais complexos, utilizando o modelo de carro disponível em [6].

Na oitava semana aparece rapidamente o envio e recepção de mensagens simples via rede, implementados em objetos mestre e escravo no jogo, com os quais os estudantes são incentivados a completar um “jogo de pegar”, denominado de **Ghost**. Neste jogo, os personagens são fantasmas que podem atravessar quaisquer paredes (o jogo é plano), mas não se vêem através delas. O maior desafio é implementar o teste de fim, onde deve ser identificado que o oponente “chegou por trás”.

Na nona semana, pouco após o meio do curso, introduzimos o *engine* **Crystal Space**. A sua estrutura de código é bem mais complexa, envolvendo sistema de arquivos virtual

(VFS), sistema de *plugins*, com consultas durante tempo de execução sobre quais interfaces são implementadas pelos objetos, entre outras facilidades, as quais tornam o código extenso. Após os primeiros exemplos com a definição manual de objetos por triângulos e polígonos, na décima semana, passa-se a usar o arquivo de descrição de mapas desse sistema para descrever cenas simples e dar exemplos de animação quadro a quadro e como se as usa. A Fig. 4 ilustra o exemplo de animação das asas de um corpo utilizado no curso.

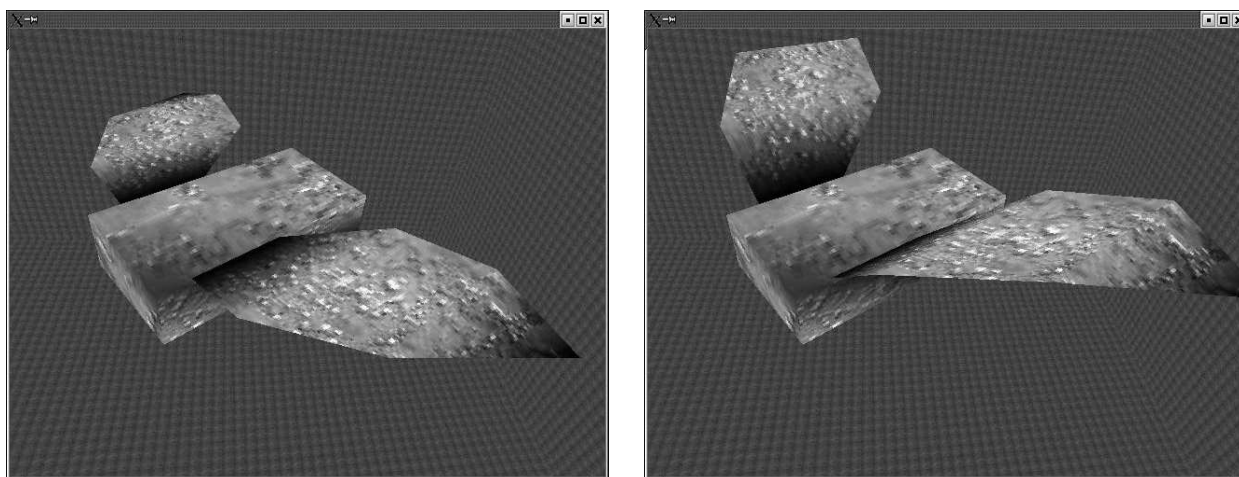


Figura 4 – Exemplo de animação de um par de asas usando os recursos de descrição de objetos no arquivo de mapas do Crystal Space.

Na décima primeira semana, novamente com o sistema Gizmo3D, se trabalha com a representação de terrenos, já vista antes no exemplo **Drive**, mas agora descrita por uma função paramétrica, permitindo ajuste dos níveis de detalhe. Também se utiliza uma função que retorna a altura de forma contínua, de acordo com um mapa de alturas e a convenção de como são montados os triângulos, permitindo que um personagem acompanhe o terreno.

Os testes de colisão simples que vinham sendo utilizados até então também voltam à tona com o estudo do sistema de colisões do Gizmo3D na décima segunda semana. São feitos diversos testes mostrando a dificuldade de fazer principalmente o tratamento das colisões, ou seja, as alterações no movimento pretendido, na orientação do móvel que colidiu, e nos seus parâmetros (quando controlamos posição, velocidade e aceleração). Isso dá margem para perceber a necessidade de sistemas de modelagem de física.

As quatro semanas finais do curso previam o estudo de efeitos diversos, sistemas de física, superfícies curvas (rapidamente) e uma discussão a respeito de mercado, objetivos, com o planejamento e esboço de um jogo, agora considerando tudo que foi estudado. Entretanto, optou-se por fazer algumas alterações nessa última parte, de forma a concluir melhor o estudo. Será feita uma revisão de todos os conceitos, incluindo as estruturas da linguagem C++ apresentadas, definindo-se também quais elementos não foram cobertos, tanto da parte dos jogos, como da linguagem C++.

Assim, o curso se conclui com o domínio das técnicas principais e a correta caracterização dos limites desse conhecimento, permitindo que os estudantes saibam como a interface com outros elementos deve ser esperada. As alterações não excluem completamente os últimos assuntos, mas apenas lhes conferem nova ênfase, sem exigir que os alunos dominem na prática essas técnicas.

Quanto à linguagem C++, não foram consideradas as suas características mais intrincadas. A orientação que foi conferida ao curso era de formar uma base segura de programação, com o pleno domínio dos recursos mais importantes que a tornam uma linguagem orientada a objetos, obtendo produtividade em primeiro lugar. Para isso, nem todos os mecanismos da linguagem são necessários. Os tópicos abordados foram selecionados pela sua importância na linguagem e também pela sua necessidade prática em geral e nesse curso. Assim, foi apresentado o uso de *templates*, por exemplo, e as características de funções virtuais e classes abstratas, mas não foram tratadas: herança múltipla e seus problemas associados, classes amigas, particularidade das referências, sobrecarga de operadores, definição de *templates*, entre tantos outros tópicos. Os autores acreditam que essa abordagem seja importante devido à grande dificuldade que estudantes principiantes têm de dominar uma nova linguagem, e principalmente um novo paradigma de programação.

8 Resultados e Dificuldades Encontradas

A realização desse curso com todas as aulas práticas foi um desafio, mas muito recompensador. Como é de se esperar, não se dispunha de tempo suficiente previamente para montar todos os exemplos, e, de fato, era desejável que eles fossem adaptados passo a passo para que houvesse continuidade e os problemas fossem justificados. Por essa razão, antes de cada aula uma série de desafios aparecia até que cada técnica fosse dominada.

A principal dificuldade encontrada, genericamente, disse respeito à falta de documentação em ambos os sistemas utilizados. No Gizmo3D, há uma documentação que apresenta o que ele tem, alguns conceitos, e alguns exemplos de código (rodando) que foram indispensáveis para o domínio do ambiente. Estudando-os, extraíram-se as características fundamentais, e se pode elaborar quase todos os exemplos pretendidos. Entretanto, não há qualquer documentação detalhada sobre os objetos e seus métodos, seja em texto seja no código fonte. Assim, a maior parte dos detalhes foram descobertos por experimentação.

Pode-se citar como exemplos de dificuldades específicas: a orientação obrigatória dos vértices de um triângulo para que uma face seja visível ou reflita luz (Gizmo3D e Crystal Space); não poder especificar o intervalo de tempo utilizado por uma possível chamada da rotina *OnTick* quando se usa na verdade a rotina *OnIdle* (Gizmo3D); impossibilidade de fazer um grafo com ciclos na hierarquia do que se chama *Scene Graph* (Gizmo3D), mas apenas árvores; diversas rotinas que deviam ser chamadas em alguma ordem (Gizmo3D e Crystal) e não em outra; falta de tratamento adequado de exceções na prática, fazendo com que o sistema travasse quando estávamos deixando de definir algo, em vez de emitir uma mensagem de erro (Gizmo3D e Crystal Space); entre diversas outras.

No Crystal Space, há uma documentação bem mais abundante. Ainda assim, na parte dos tutoriais, faltam exemplos simples para algumas das técnicas. Por exemplo, para exemplificar um sistema de física, há centenas de linhas de código que tratam de colisão, o que aparentemente não tem efeito nenhum no exemplo, e não é devidamente explicado nos manuais. Assim, tornou-se impraticável dar exemplos concretos com o sistema de física em tempo hábil. A animação por esqueleto também possui uma quantidade de detalhes muito grande, e, se fosse de interesse do curso, também não poderia ter sido coberta sem um exemplo mais didático.

Há um conjunto de técnicas e funções mais simples que não foram cobertas simplesmente por falta de tempo, como usar domos, nuvens e iluminação para modelar melhor o céu, utilizar qualquer tipo de som, experimentar e comparar técnicas de iluminação estática, dinâmica, usar texturas procedurais, sistemas de partículas, sombras, entre tantas outras. Espera-se que em outras edições do curso algumas destas técnicas possam ser também experimentadas na prática com código adequado.

Outra dificuldade encontrada que deve ser considerada na realização do curso está na dedicação que demanda por parte dos alunos. O contexto onde esse curso foi aplicado pela primeira vez continha um conjunto pequeno de alunos com condições de cursá-lo, mas que acumulavam ou atividade profissional, ou bolsa de pesquisa, ou trabalhos de conclusão de curso, além das cadeiras e trabalhos do Curso de Ciência da Computação em que estão matriculados (na sua maioria). Nesse contexto, não é possível cobrar grande dedicação extra classe dos alunos, o que dificulta que eles dominem todas as técnicas. O caráter prático garante que eles compreendam as técnicas e as experimentem um pouco pela alteração no código. Mesmo assim, eles não terão o mesmo domínio que atingiriam se pudessem dispensar no mínimo mais uma hora diária com os exemplos das aulas ou testando seus próprios exemplos e idéias. Por esta razão, recomenda-se considerar esse fator na realização do curso e deixar claro para os estudantes que o nível de envolvimento irá influir no domínio final do conteúdo coberto.

9 Conclusões

Este trabalho apresentou os aspectos fundamentais envolvidos na realização de um curso prático de programação em C++ para jogos 3D, o qual pode ser aplicado tanto em cursos que visam à formação de programadores de jogos como em cursos regulares de graduação em computação, com o objetivo primeiro de ensinar a programação em linguagem C++.

Pelo seu caráter prático, o curso exige considerável dedicação tanto dos professores, na preparação dos exemplos e prática com os motores, quanto dos alunos, que também terão tanto maior aprendizado quanto mais tempo tiverem disponível para experimentar e desenvolver seus próprios exemplos. Uma das principais vantagens da abordagem utilizada é que os estudantes estão continuamente expostos a técnicas profissionais de programação, isto é, com problemas e bibliotecas de *software* reais, e não apenas com “*toy programs*”, apesar de isso parecer paradoxal por se tratar de jogos. E o caráter prático do curso garante que os estudantes não apenas são apresentados a esses sistemas, como estão modificando código completo (que roda) durante todas as aulas.

Além do caráter prático, ressaltamos a importância da metodologia construtiva que foi utilizada, onde no início apenas são usados objetos prontos, destacando a vantagem do mecanismo de orientação a objetos no tocante à produtividade. Somente após algum tempo é introduzida a definição de classes, a qual também é justificada pela simplicidade que traz na programação da parte principal dos exemplos, que deve ser maior do que a dificuldade de fazer as definições. Outra parte essencial da metodologia também é a idéia de somente introduzir os mecanismos fundamentais da linguagem e explorar a produtividade com eles. Assim, acredita-se que a exclusão proposital da maior parte das exceções e dos intrincados mecanismos de C++ facilita o domínio e a aceitação da linguagem por parte dos alunos, sem trazer, até onde foi considerado, lacunas à sua formação.

Espera-se que essa experiência sirva de estímulo para outros cursos, e que alguns dos problemas identificados sejam locais e plenamente resolvidos em outras edições. Os autores colocam-se à disposição para compartilharem experiências semelhantes e trocar informações técnicas sobre o assunto.

10 References

1. Bjarne Stroustrup. C++ Programming Language. Reading: Addison Wesley. 1997.
2. Gamma, E., Helm, R., Johnson, R., Vlissides, J. Design Patterns – elements of reusable object-oriented software. Reading: Addison Wesley, 1995.
3. ToolTech Software. Gizmo3D, <http://www.tooltech-software.com> (21/08/2002).
4. Tyberghein, J., Sunshine, E. (Adm.). Crystal Space <http://sourceforge.net/projects/crystal> (21/08/2002).
5. Crawford, Chris. The Art of Computer Game Design. Washington State University at Vancouver - Department of History, 1982.
6. Tom Greenway. Car 1. Disponível em: <http://www.3dtotal.com> (29/11/2002).