Universidade Federal de Mato Grosso do Sul
Faculdade de Computação

Computação Gráfica 2019

Trabalho – Parte 1

Paulo A. Pagliosa

O objetivo da **Parte 1** é projetar e implementar o modelo de cena e seus elementos a ser usado nas partes subsequentes do trabalho. No Capítulo 1, vimos que, conceitualmente, uma cena é uma coleção de atores e de luzes. Neste exercício, inspirado em motores de jogos tais como o Unity[1], é proposto um modelo de cena que permite a construção de hierarquias de objetos em que cada objeto de cena pode ser composto por componentes de vários tipos, entre os quais a geometria dos atores e as luzes da cena.

1 Passo 1: Código-fonte e geração do executável

O código-fonte **específico** para este exercício está disponível no diretório `cg/p1` do repositório `https://git.facom.ufms.br/pagliosa/cg.git`.

Observação: o código comum a todas as partes do trabalho poderá ser alterado entre uma e outra parte do trabalho. Assim, antes de obter o código específico de algum exercício, atualize o código comum a partir do diretório `cg/common` do `git`.

Após atualizar a parte comum e baixar a parte específica em seu diretório de trabalho, `cg` conterá os diretórios `common`, `p0` (com o seu código da **Parte 0**) e `p1`. O conteúdo de `p1` será:

```
assets
  p1.fs
  p1.vs
build
  vs2019
    imgui.ini
    p1.sln
    p1.vcxproj
    p1.vcxproj.filters
Component.h
imgui_demo.cpp
Main.cpp
P1.cpp
P1.h
Primitive.h
Scene.h
SceneNode.cpp
SceneNode.h
SceneObject.cpp
```

[1] `https://unity3d.com/pt`.

```
SceneObject.h
Transform.cpp
Transform.h
```

Como no exercício anterior, o diretório `assets` contém o código dos *shaders* de vértice e fragmento, e `build/vs2019` o projeto do Visual Studio 2019. O arquivo `imgui_demo.cpp` contém exemplos de todos os elementos gráficos da ImGui e pode ser usado, se necessário, como referência durante o desenvolvimento do exercício. A definição e implementação da classe de janela gráfica da aplicação são codificadas, respectivamente, em `P1.h` e `P1.cpp`.

No Visual Studio 2019, abra o arquivo `build/vs2019/p1.sln` e construa o arquivo executável (você pode fazer isso na configuração de *debug* enquanto desenvolve o exercício, mas não se esqueça que, ao final, o programa executável a ser entregue deve ser construído na configuração de *release*). Você notará, entre outros, o arquivo `p1.exe` no diretório `p1`. A execução do programa exibe em sua tela uma janela gráfica com um retângulo em cores (na verdade, uma das faces de uma caixa) e uma interface com o usuário com duas janelas, como na Figura 1. A demonstração das funcionalidades da interface foi feita em sala.

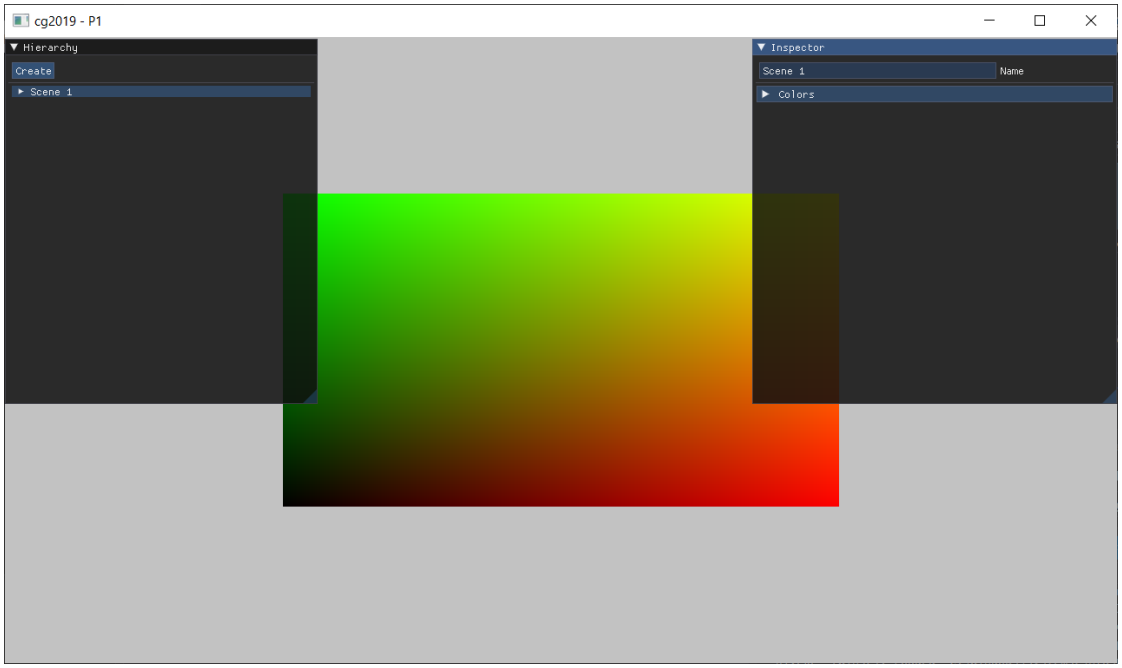

Figura 1: Programa `p1.exe` (no Windows).

2 Passo 2: Tarefas

No modelo proposto, uma *cena* contém uma coleção de *objetos de cena*, representados pelas classes `Scene` (em `Scene.h`) e `SceneObject` (em `SceneObject.h`), respectivamente. Um objeto de cena, O, também pode conter uma coleção de outros objetos de cena, os *filhos* de O. O objeto de cena O é *pai* dos objetos de cena nele contidos. Um objeto de cena conhece seu objeto de cena pai e também a cena em cuja hierarquia está contido. Os objetos de cena sem pai são os objetos de cena da *raiz* da cena.

Um objeto de cena contém ainda uma coleção de *componentes*, os quais encapsulam propriedades do objeto de cena. Todo objeto de cena (mesmo um objeto de cena vazio) contém um componente do tipo `Transform` (definido em `Transform.h`), o qual especifica

a posição, orientação e escala do objeto de cena. Um objeto de cena pode conter outros tipos de componentes. Neste exercício, a caixa exibida na Figura 1 é um objeto de cena que contém um componente do tipo `Primitive` (definido em `Primitive.h`), o qual representa a geometria da caixa. Todo objeto de cena que tem uma forma (a ser renderizada) deve ter pelo menos um componente do tipo `Primitive`. Você e eu estenderemos `Transform` e `Primitive` e desenvolveremos outras classes de componentes nos próximos exercícios.

As classes `Scene` e `SceneObject` derivam de `SceneNode` (em `SceneNode.h`). Um nó de cena contém um nome e pode ser exibido como um nó de árvore na janela `Hierarchy` da interface gráfica. O nó de cena correntemente selecionado, ou *nó corrente*, tem suas propriedades exibidas na janela `Inspector` da interface gráfica. Quando o nó corrente é um objeto de cena, seus componentes são mostrados em `Inspector`. Assim, para cada tipo de componente, deverá haver uma função que exibe em `Inspector` os elementos de interface para manipulação das propriedades do componente.

As atividades de programação dessa parte do trabalho consistem em:

A1 Projetar e implementar a coleção de objetos de cena da hierarquia de objetos de cena de uma cena. Você pode adotar, por exemplo, uma lista duplamente encadeada para implementar a coleção de objetos de cena de uma cena bem como de outros objetos de cena. Seja qual for sua escolha, você deverá justificá-la em seu arquivo `README`. Sua estrutura de dados deve permitir a adição e remoção de objetos de cena, bem como prover um iterador para seu atravessamento, pelo menos.

A2 Implementar, em `SceneObject.cpp`, `SceneObject::setParent(parent)`, método que define o objeto de cena pai de um objeto de cena. Se `parent == nullptr`, o objeto de cena não terá pai, isto é, fará parte da raiz da cena. Caso contrário, se este já não fizer parte da raiz da cena, será removido da coleção de filhos de seu pai corrente e, então, adicionado à coleção de filhos de `*parent`, o qual passará a ser seu novo pai.

A3 Projetar e implementar a coleção de componentes pertencentes a um objeto de cena (a qual conterá, pelo menos, o componente `Transform` do objeto de cena). Assim como em **A1**, você deverá justificar sua escolha em seu arquivo `README`. Sua estrutura de dados deve permitir a adição e remoção de componentes, bem como prover um iterador para seu atravessamento, pelo menos.

A4 Alterar, em `P1.cpp`, o método `P1::buildScene()`, responsável por construir a cena inicial da aplicação. A versão corrente cria uma cena chamada `Scene 1` e um único objeto de cena chamado `Box 1` (que deveria ser um objeto de cena da raiz da cena), além de um primitivo para a geometria da caixa (que deveria ser um componente de `Box 1`). Adicione, na cena que você criar, vários objetos de cena (vazios, isto é, sem outros componentes além de `Transform`, e/ou caixas como `Box 1`), tanto na raiz da cena como na coleção de filhos de outros objetos de cena. A hierarquia criada deverá ter pelo menos 2 níveis (a raiz da cena está no nível zero).

A5 Alterar, em `P1.cpp`, o método `P1::hierarchyWindow()`, a fim de que a hierarquia de suas cenas seja exibida na interface gráfica. Ainda, implementar a funcionalidade do botão `Create`, para adicionar ao nó corrente um novo objeto de cena, que pode ser, nesse exercício, vazio ou com a geometria de uma caixa, como `Box 1`. Objetos de cena vazios devem ser nomeados `Object 1`, `Object 2` e assim por diante. Caixas devem ser nomeadas de maneira similar, isto é, `Box 2`, `Box 3`, etc.

A6 Alterar, em `P1.cpp`, o método `P1::sceneObjectGui()`, responsável por exibir na janela `Inspector` da interface gráfica os componentes do objeto de cena correspondente ao nó corrente (o método já mostra o componente `Transform`). Cada tipo diferente de componente deve ter uma função própria que o exibe em `Inspector`.

A7 Alterar, em `P1.cpp`, o método `P1::render()`, a fim de que todos os primitivos (de objetos de cena) da cena sejam renderizados e, assim, exibidos na janela da aplicação.

3 Passo 3: README

Como no exercício anterior, o arquivo `README` deve conter o nome(s) do(s) autor(es) e uma descrição de como gerar (caso o Visual Studio 2019 não tenha sido utilizado) e executar o programa. Em seguida, você deve descrever quais as atividades você conseguiu ou não implementar, parcial ou totalmente, além de um breve manual do usuário (por exemplo, se é preciso usar alguma tecla para alguma funcionalidade para a qual não há ajuda textual na interface gráfica com o usuário). Descreva também quaisquer outras extensões que você implementou por iniciativa própria. (Por exemplo, você poderia sobrescrever, na classe `P1`, o método `keyInputEvent()`, herdado de `GLWindow`, para tratamento de uma tecla para remoção, da hierarquia de objetos de cena, do objeto de cena correspondente ao nó corrente). Estas podem valer pontuação extra em sua nota do exercício. Na descrição das atividades **A1** e **A3**, você deve justificar as estruturas de dados que você adotou no projeto das classes de cena e de objetos de cena.

4 Passo 4: Entrega do programa

O exercício deve ser entregue via AVA em arquivo único compactado (somente um arquivo por grupo), chamado `p1.zip` (nome(s) do(s) autor(es) vão no arquivo `README`), contendo:

- o diretório `p1 com` o código-fonte completo e com o arquivo executável `p1.exe`, mas **sem** quaisquer arquivos intermediários de backup ou resultantes da compilação, tanto em `p1` como em seus subdiretórios; e
- o arquivo `README`.

Não serão considerados e terão nota zero programas plagiados de qualquer que seja a fonte, mesmo que parcialmente, exceção feita ao código fornecido pelo professor.

5 Lista de objetivos

A verificação dos objetivos da **Parte 1** levará em conta se:

☐ O arquivo `p1.zip` foi submetido com os arquivos corretos, e o arquivo `README` contém as informações solicitadas.

☐ O projeto da coleção de objetos de cena de uma cena e de outros objetos de cena foi justificado e implementado, e a implementação está correta.

☐ O método que define o objeto de cena pai de um objeto de cena foi implementado, e a implementação está correta.

☐ O projeto da coleção de componentes de um objeto de cena foi justificado e implementado, e a implementação está correta.

☐ O método de criação da cena inicial foi usado efetivamente para testar as estruturas de dados e métodos especificados em **A1**, **A2** e **A3**, e funciona conforme descrito em **A4**.

☐ A interface gráfica com o usuário exibe corretamente a hierarquia de objetos de cena da cena.

☐ O botão `Create` permite a criação de objetos de cena conforme descrito em **A5**.

☐ A interface gráfica com o usuário exibe corretamente os componentes do objeto de cena correspondente ao nó corrente.

☐ Todos os primitivos da cena são renderizados e exibidos.