MC937A/MO603A – Computação Gráfica - 2021-S2 - Jorge Stolfi
Trabalho de laboratório 02 - 2021-08-25
Batalha interplanetária

Objetivos: treinar uso de comandos da linguagem POV-Ray: repetição, testes condicionais, e procedimentos ("macros").

Enunciado. Seria ingênuo pensar que visitantes extraterrestres, se e quando eles aparecerem, serão pacíficos e amigáveis. Todos nós temos que nos preparar para uma invasão em massa. Uma maneira eficaz de fazer isso é simular tal invasão, e a resposta terráquea, usando POV-Ray. Esta será sua tarefa de hoje.

Especialistas em psicologia de alienígenas garantem que, em batalha, eles prezam acima de tudo a rígida formação de naves e exércitos. Então, na sua simulação, tanto a armada invasora quanto nosso pelotão ou esquadra deverão estar dispoastas em uma grade regular de algum tipo. Por outro lado, durante a batalha, naturalmente haverá lacunas nessa grade devido à destruição de naves, tanques, aviões, skateboards, etc.. Felizmente o POV-Ray, nossa arma secreta, dispõe de comandos de repetiçao e condicionais que permitem fazer isso.

Parte 1. Desenhe, à mão livre, um esboço dos dois exércitos e mostre o desenho para o professor. Cada exército deve ser uma grade regular de itens similares, com algumas lacunas. Cada item deve ser formado por pelo menos três objetos primitivos do POV-Ray.

Parte 2. Apenas depois que o professor aprovar seu esboço: Copie os arquivos da aula passada para uma nova sub-pasta `2021-08-25` da pasta `mc937` no seu computador. Edite o arquivo `main.pov`, trocando a cena da aula anterior pela descrição das duas armadas. Use o comando `make` para produzir uma imagem do mesmo.

Cada armada deve ser modelada como uma macro com dois parâmetros que definem o número de fileiras e colunas da mesma. Cada item deve deve ser uma macro com pelo menos um parâmetro que permite aplicar pequenas variações em alguma característica do mesmo, como cor, tamanho, pontaria, etc.. Cada armada deve ser gerada por duas malhas de repetição (`while`) encaixadas, com comandos condicionais (`if`) para obter as lacunas.

Exportação. Não se esqueça de exportar seu arquivo `main.pov` até o final da aula para sua pasta WWW pública

`http://students.ic.unicamp.br/~ra`*SEU_RA*`/mc937-2021-2/2021-08-25/`

Originalidade. O arquivo de descrição `main.pov` deve ser construído manualmente, com um editor de texto comum, **sem** o auxílio de qualquer editor gráfico ou outra ferramenta de modelagem geométrica. Não é permitido copiar ou incluir quaisquer arquivos POV-Ray além dos fornecidos pelo professor ou escritos por você mesmo. Porém, é permitido re-usar arquivos ou trechos de código de exercícios anteriores.

Individualidade. Lembre-se de que todo trabalho prático é **individual**. Não é permitido pedir qualquer tipo de ajuda a colegas ou outras pessoas. Dúvidas devem ser tiradas apenas com o professor.

(continua)

Comandos. Os comandos POV-Ray necessários para este exercício são:

- Definição de macro:
  `#macro` *nome* `(`*parâmetros*`)`
  comandos
  `#end`

- Declaração e atribuição de variáveis locais dentro de uma macro:
  `#local` *nome* `=` *número ou vetor*`;`
  `#local` *nome* `=` *objeto*

  O valor pode ser uma fórmula. Note que o ponto-e-vírgula é obrigatório se o valor for um número ou vetor, e proibido se o valor for um objeto ou textura. Note também que a chave `#local` é obrigatória inclusive para alterar o valor de uma variável local já definida. O comando `#declare` é semelhante mas define ou altera variáveis globais.
  Atenção: Os seguintes nomes são palavras reservadas da linguagem POV-Ray: `x`, `y`, `z`, `u`, `v`, `t`.

- Retorno de resultado de uma macro: Não há comando `return` explícito. O resultado da macro é qualquer fórmula ou objeto que aparecer sem o `#local` ou `#declare`. Por exemplo:

```
#macro treco(tipo)
  #local R = 10*sqrt(2);
  union{
    sphere{ <0,0,0>, R }
    #if (tipo = 7)
      cylinder{ <0,0,R>, <0,0,2*R>, 5 }
      box{ <-10-10,2*R>, <+10,+10,2*R+10> }
    #else
      cone{ <0,0,R>, R/2, <0,0,3*R+10> 1.5*R }
    #end
  }
#end
```

  O resultado da macro será `union{ .. }`.

- Invocação de uma macro:
  nome`(`*argumentos*`)`
  Por exemplo:

```
#local raio = calcula_raio(...);
object{ treco(tt) translate <10,10,0> }
```

  A segunda forma deve ser usada quando o resultado da macro for um objeto (simples ou composto). Se o resultado for dois ou mais objetos, eles devem ser agrupados com `union{ ... }` dentro da macro.

- Comando condicional:
  `#if (`*condição*`)`
  comandos
  `#else`
  comandos
  `#end`

  A *condicão* pode usar operadores de comparação `=` (note: não `==`), `!=`, `>`, `<`, `>=`, `<=`, ou operadores lógicos `&` ("e"), `|` ("ou"), `!` ("não"). A parte `#else` *comandos* pode ser omitida.

- Comando de repetição:
  `#while (`*condição*`)`
  comandos
  `#end`

  Este comando funciona como o similar em C e outras linguagens. Por exemplo:

```
union{
  #local kk = 0;
  #while (kk < N)
    sphere{ <0, 0, kk*kk >, 5 translate < 0, kk*10, 0> }
    #local kk = kk+1;
  #end
}
```

  Note que é necessário incrementar explicitamente a variável contadora `kk` com o comando `#local kk = kk+1`.