

Relatório: Desenvolvimento de uma aplicação gráfica interativa

Matheus Leite Cruz

01 de Dezembro de 2019

Sumário

1	Introdução	2
2	Instalação e organização dos arquivos	2
2.1	Instalação e execução	2
3	Implementação da Aplicação	2
3.1	Organização dos módulos	2
3.2	Main	4
3.3	Models	4
3.3.1	matrix	4
3.3.2	load_textures	5
3.3.3	obj_model	5
3.3.4	composite_object	6
3.3.5	complex_object	6
3.3.6	scene_obj	6
3.4	Shader	7
3.4.1	shader_program	7
3.5	World	8
3.5.1	lighting	8
3.5.2	free_camera	8
3.5.3	view	8
3.6	game_loop	9
3.6.1	Estado do jogo	9
3.6.2	Inicialização	10
3.6.3	loop principal	10
4	Funcionamento da Aplicação	12
4.1	Controles	12
4.2	Funcionamento	13
5	Requisitos	18

1 Introdução

O trabalho consiste no desenvolvimento de uma aplicação gráfica interativa. Foi desenvolvido um jogo simples que permite que o usuário controle um "personagem" principal e colete objetos espalhados sobre um plano, adquirindo pontos. Conforme o "jogador" adquire pontos, a fidelidade gráfica da aplicação é melhorada, seguindo uma progressão lógica de modelos de iluminação, texturas e controle/perspectiva da câmera.

A programa foi desenvolvido na linguagem de programação rust(1), utilizando OpenGL. A escolha de linguagem foi motivada pelo seu potencial para o desenvolvimento de aplicações gráficas complexas, mesmo com um "ecossistema" ainda um pouco imaturo para desenvolvimento de jogos(2).

2 Instalação e organização dos arquivos

2.1 Instalação e execução

Para executar o projeto:

1. Extrair o arquivo zip, abrir a pasta bin e executar o arquivo upgrade.exe ou upgrade_low_tex.exe (poucas texturas para carregamento mais rápido). É necessário manter a estrutura de pastas original.

Para compilar e executar o projeto:

1. Instalar as dependências da linguagem rust.
 - 1.1 Acessar <https://www.rust-lang.org/tools/install> e seguir as instruções para sua plataforma
 - 1.2 Instalar a versão stable com o instalador do passo anterior.
2. Extrair o zip do projeto em alguma pasta ou utilizar o git para clonar o projeto (disponível em https://github.com/mlcruz/upgrade_rs)
3. Executar o comando 'cargo run' na raiz da pasta. Certifique-se que a pasta dos binários do rust esteja no PATH
 - 3.1 Ao executar o comando cargo run, todas as dependências vão ser baixadas e o projeto será compilado.

3 Implementação da Aplicação

3.1 Organização dos módulos

Todos os arquivos de recursos necessários ao programa em tempo de execução estão na pasta data. As pastas models, shaders e world contem módulos que implementam as funcionalidades do jogo. Cada modulo é composto por uma

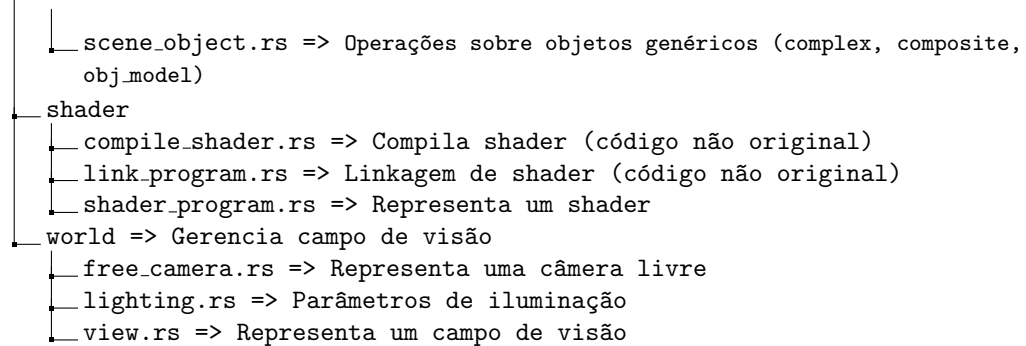
lista de arquivos de extensão .rs, e um arquivo mod.rs. Módulos tem o objetivo de expor funcionalidades associadas e organizar o código.

Os arquivos de código estão documentados de maneira que o nome de suas variáveis e funções sejam descritivos, e comentários redundantes foram evitados. De maneira geral, se o nome da função é (por exemplo) dot_product, não existirá um comentário descrevendo seu funcionamento ou implementação, sendo assumido que a implementação é clara sem maiores comentários. A navegação pelo código fonte funciona melhor se utilizado algum editor que possibilite links entre funções e suas definições, como o VS code.

Boa parte do código tem base no que foi utilizado nos laboratórios, mas algumas outras tiveram que ser reimplementadas (por exemplo, carregamento de arquivos obj e texturas). No geral a falta de algumas bibliotecas causou uma certa dificuldade na realização do projeto, mas na medida do possível todos os requisitos devem ter sido atingidos.

Omitindo o arquivo mod.rs:

```
|
├─ Cargo.toml => Arquivo para gerenciamento de dependências
├─ Cargo.lock
├─ src
│   ├── game_loop.rs => Loop principal de lógica do jogo
│   ├── handle_input.rs => Tratamento de entrada do usuário
│   ├── main.rs => Inicialização de contexto
│   └─ data
│       ├── shader => Shaders utilizados
│       │   ├── fragment
│       │   │   ├── blinn_phong_ilumination.glsl
│       │   │   ├── default.glsl
│       │   │   ├── gouraud_fragment_lambert.glsl
│       │   │   ├── gouraud_fragment_phong.glsl
│       │   │   ├── lambert_ilumination.glsl
│       │   │   └─ phong_ilumination.glsl
│       │   └─ vertex
│       │       ├── default.glsl
│       │       ├── gouraud_shading_lambert.glsl
│       │       ├── gouraud_shading_phong.glsl
│       │       └─ phong_shading.glsl
│       ├── objs => Pasta contendo arquivos obj
│       ├── textures => Pasta contendo texturas
│       └─ models => Gerencia criação e renderização de objetos
│           ├── complex_obj.rs => Objeto recursivos
│           ├── composite_obj.rs => Objetos simples com n filhos
│           ├── load_texture.rs => Carregamento de texturas
│           ├── matrix.rs => Transformações de matrix
│           └─ obj_model.rs => Representa um Objeto simples base
```



3.2 Main

A função principal do programa tem o objetivo de carregar os ponteiros para as funções do OpenGL, inicializar o contexto e a janela, e após isso iniciar o loop principal do jogo.

Para a criação de contexto foi utilizada a biblioteca glutin(3), uma alternativa ao glfw nativa ao rust (sem precisar compilar o próprio glfw externamente).

Para carregar os ponteiros para o OpenGL foi utilizada a biblioteca gl-rs (4), alternativa a biblioteca glad

3.3 Models

3.3.1 matrix

Foi utilizada a biblioteca glm para definir estruturas de dados e realizar operações simples entre vetores e matrizes. Foram implementadas funções para:

- Rotações (x,y,z, eixo) de matriz 4x4
- Translações de matriz 4x4
- Translação para 0 => Operação de rotação ou escala => Translação para posição original
- Escala de matriz 4x4
- Calculo de produto escalar e vetorial
- Calculo de matriz de projeção ortográfica
- Normalização de vetores
- Matriz de camera view

As funções implementadas foram uma praticamente "tradução literal" para rust das nos laboratórios implementadas em c++.

Todos os Objetos (SceneObject, ComplexObj, ObjModel, CompositeObj), implementam funções de transformações matriciais derivadas das acima. A maior todos as outras estruturas utilizam funções definidas nessa biblioteca de uma maneira ou outra.

3.3.2 load_textures

Gerencia carregamento texturas para a gpu a partir de imagens. Retorna um u32 representando a textura carregada.

3.3.3 obj_model

Estrutura que representa um objeto a ser desenhado. Utiliza a biblioteca tobj - tiny obj loader(5) para fazer o carregamento dos arquivos obj

Atributos: Um objeto é inicializado a partir de um arquivo .obj, definindo diversas propriedades a respeito de como deve ser desenhado na tela:

- vao: Endereço da vao do objeto, não é criada em clones.
- ebo: Indices do obj.
- texture, normal, geometry vbo: Vbos com dados de textura, normal e geometria. Normais são calculadas a partir dos vértices do obj, se não existirem.
- model: Matriz model inicial do obj. Padrão é matriz identidade.
- index_len: Tamanho do do vetor de índice dos vértices
- bbox_min/max: Bounding box computada na inicialização
- lighting_source_override: vetor de sobrescrita do local da fonte de luz
- color_override: Vetor de sobrescrita da refletância
- ambient_reflectance_override: Vetor de sobrescrita da refletância ambiente
- specular_reflectance_override: Vetor de sobrescrita da refletância especular
- phong_q: Vetor de sobrescrita da refletância especular
- Texture override: Local da textura que sobrescreve a textura atual se texture map type for setado.
- Textura map type: Tipo de mapeamento da textura.
 - 0 - Arquivo OBJ
 - 1- Planar XY
 - 2 -Planar XY
 - 3- Esférico
 - 4- Cilíndrico

Objetos copiados a partir de um objeto não realocam vaos e vbos, somente seus parâmetros enviados para a GPU como uniforms diferem. A maior parte das funções sobre objetos é imutável, e quase sempre retorna um objeto novo com as alterações aplicadas sem alterar o objeto base que chama o método.

Metodos: Um objeto implementa as seguintes funções (além de get/set/with de seus atributos, que são implementados para quase todos e transformações matriciais):

- new: Inicializa objeto, atribuindo valores padrão aos atributos, carregando arquivo obj especificado, computando normais e inicializando vaos e vbos na GPU.
- load_texture: Retorna objeto a partir do obj base com a textura especificada carregada em seu override a partir de um arquivo.
- check_intersection: Detecta interseção entre obj e alguma outra bounding box.
- draw: Carrega uniforms da GPU com atributos do obj, e desenha na tela. Antes de desenhá-lo, todas as propriedades relevantes do mesmo (matriz model, bounding_box e etc) são enviadas para a gpu.

3.3.4 composite_object

Estrutura que representa um ObjModel raiz com diversos filhos. É Criado quando um SceneObject que é um ObjModel chama o método add_children em outro ObjModel.

Métodos: Implementa add_children, draw e operações de transformação matricial. Transformações aplicadas a raiz são aplicadas a todos os filhos, e draw desenha todos os filhos com as transformações aplicadas. É inicializado a partir de um add_children de outro objeto

3.3.5 complex_object

Estrutura que representa um ObjModel raiz com diversos filhos que podem ser outros ComplexObj. É Criado quando um CompositeObj ou ComplexObj chama o método add_children em outro CompositeObj/ComplexObj. É inicializado a partir de um add_children de outro objeto.

Métodos: Implementa add_children, draw e operações de transformação matricial. Transformações aplicadas a raiz são aplicadas a todos os filhos, recursivamente, e draw desenha todos os filhos recursivamente.

3.3.6 scene_obj

Objeto do tipo ComplexObj OU ObjModel OU CompositeObj (tipo algébrico soma). Implementa operações genéricas que funcionam sobre qualquer um dos

3 objetos. No caso de objetos que contem raiz, todas as transformações são aplicadas a raiz e seus filhos recursivamente.

Todas as detecções de colisão são verificadas recursivamente para filhos, se existirem.

Métodos: Além de implementar get, set, with sobre atributos dos objetos base, add_children e transformações matriciais:

- new: Inicializa um ObjModel novo com o arquivo .obj especificado
- get_root: Retorna raiz do Objeto. É ele mesmo para ObjModels
- check_plane_intersection: Detecta interseção entre objeto e algum plano.
- check_is_intersecting_fence: Detecta interseção entre objeto e alguma cerca bi-dimensional.
- check_point_intersection: Detecta a intersecção entre o objeto e um ponto
- check_bbox_intersection: Detecta a intersecção entre o objeto e uma bounding box
- draw: Carrega uniforms e desenha recursivamente (para obj e cada filho).

3.4 Shader

3.4.1 shader_program

Estrutura que representa o programa de um shader compilado e linkado. Os arquivos de fonte de cada shader podem ser encontrados em /src/data/shader. Objetos utilizam um shader para desenhar na tela.

Fragment shaders implementados (em /src/data/shader/fragment):

- Iluminação de blinn-phong, phong shading => blinn-phong_ilumination.glsl
- Iluminação de phong, phong shading => phong_ilumination.glsl
- Iluminação de lambert, phong shading => lambert_ilumination.glsl
- Iluminação de phong, gouraud shading => gouraud_fragment-phong.glsl
- Iluminação de lambert, gouraud shading => gouraud_fragment_lambert.glsl
- Cor sem iluminação=> default.glsl

Vertex shaders implementados (em /src/data/shader/vertex):

- Calcula normal, posição => default.glsl
- Iluminação de phong, gouraud shading => gouraud_shading-phong.glsl
- Iluminação de lambert, gouraud shading => gouraud_shading_lambert.glsl
- phong shading => phong_shading.glsl

3.5 World

3.5.1 lighting

Estrutura que representa uma configuração de iluminação da cena.

Atributos: Vetores de iluminação global, Iluminação ambiente e Direção da iluminação.

3.5.2 free_camera

Estrutura que representa uma câmera na cena. Tem todos os atributos necessários para calcular sua própria view matrix, e depois é utilizada pela estrutura View para montar a cena (carregando as matrizes de projeção e view) antes da renderização.

Atributos:

- pos: Posição da câmera
- distance: distância da câmera
- pitch : Movimento angular vertical sobre o eixo fixo
- yaw : Movimento angular lateral sobre o eixo fixo
- view_matrix: view matrix calculada a partir dos outros atributos da câmera
- front: Vetor da direção em que a câmera aponta

Métodos:

- new: Iniciliza uma câmera a partir de uma posição, e valores de pitch e yaw.
- refresh: calcula view matrix da câmera a partir de sua posição e distância, sem alterar o vetor front pre estabelecido. É utilizado para simular uma câmera look-at
- refresh_as_free_camera: calcula view matrix da câmera a partir de sua posição, distância, pitch e yaw.

3.5.3 view

Estrutura que representa um campo de visão, com câmera e iluminação.

Atributos:

- nearplane: plano limite mínimo do frustum do campo de visão
- farplane : plano limite máximo do frustum do campo de visão
- projection_matrix: Matriz de projeção perspectiva ou ortográfica.

- camera: FreeCamera a ser utilizada para preparar campo de visão
- lighting: Lighting a ser utilizado para calcular parâmetros de iluminação

Métodos: Além de implementar get, set, with sobre os atributos

- new: Inicializa uma View a partir de uma FreeCamera, um nearplane e um farplane. Como padrão, utiliza projeção perspectiva e parâmetros de iluminação pre-definidos.
- render: Utiliza view matrix da câmera, matriz de projeção e parâmetros de iluminação para preparar uniforms a serem utilizados pela GPU para desenhar a cena. Carrega todos os seus atributos relevantes na gpu, deve ser chamado antes de desenhar objetos.
- orthographic: Altera matrix de projeção para projeção ortográfica
- perspective: Altera matrix de projeção para projeção perspectiva

3.6 game_loop

Loop principal do jogo.

3.6.1 Estado do jogo

São Inicializas diversas variáveis de estado para controlar o estado do jogo:

- should_break: Indicador de encerramento de loop
- should_add_obj: Indicador de desenho de objeto na tela
- is_view_orto: Indicador de projeção ortográfica
- draw_queue: Fila de objetos a serem desenhados a cada loop
- score: Pontuação do usuário
- camera_height: Altura da câmera principal
- obj_plane_height: Altura do plano principal
- speed_mult: Multiplicador de velocidade do obj principal
- look_at: Vetor de direção da camera look-at
- camera_speed_mult: Multiplicador de velocidade da camera
- current_camera: Controle de camera atual
- can_fly: Indicador de permissão de movimento no eixo Y
- with_Bézier : Indicador de movimentação dos objs em curvas de Bézier .

- curr_x: f64 : Variável que varia de 0 a 1 conforme o tempo, utilizada para curvas de Bézier .
- dir: direção da variável de controle curr_x (crescente ou decrescente);
- complex_objs: Indicador de objetos complexos,
- max_framerate: Framerate máxima,
- progression_multiplier: Multiplicador de dificuldade na progressão ao adquirir pontos,
- lighting_source: Direção da fonte de luz global,

3.6.2 Inicialização

Inicialização:

1. Compila e linka shaders do jogo
2. Carrega texturas utilizadas pelo jogo
 - 2.1 Adiciona texturas carregadas em uma pool de texturas para ser utilizada pelos objs
3. Carrega objs utilizados pelo jogo
 - item 3.1 Adiciona objs carregados em uma pool, que servem como base para os objs gerados aleatoriamente do jogo
4. Inicializa câmera ortográfica look-at, atribui como câmera padrão
5. Inicializa câmera livre
6. Inicializa estado inicial do jogo (pontuação, shader atual, framerate, câmera atual, etc)
7. Inicializa View com câmera look-at inicial

3.6.3 loop principal

Após a inicialização, o jogo funciona executando sequencialmente as seguintes operações em loop:

1. Inicia contador de tempo de frame
2. Limpa tela
3. Trata eventos do usuário. Eventos do usuário podem modificar qualquer variável de estado de jogo, câmera, view, e obj principal. Controles possíveis são descritos na seção de funcionamento do programa.

4. Se `should_add_obj` é true:

- Restaura todas (ou quase todas) as variáveis de controle de estado do jogo para seus valores iniciais após a inicialização (score: 0). Isso acontece a cada iteração, e a cada iteração as transformações que levam aos próximos estados do jogo são reaplicadas dependendo da pontuação do usuário. Isso permite que seja possível retornar a estados anteriores/avançar estados dependendo unicamente da sem interferência de estados anteriores (ou quase).

- Aplica transformações aos objs e estado de jogo dependendo da pontuação do usuário. Todas as pontuações necessárias podem ser multiplicadas por um multiplicador de dificuldade:

- * (a partir de) 0 (pontos) : Prepara até 5 objetos novos, gerados a partir da pool de texturas e objs para serem inseridos na cena. As propriedades de modelo, textura, refletância ambiente, refletância especular, tamanho e cor dos objetos são geradas aleatoriamente a cada mudança de estado de `should_add_obj`, e ao fim das etapas de transformação, 1 a 5 objetos são inseridos na fila de desenho.

- * (a partir de) 0 (pontos) : Aumenta o tamanho do objeto principal, e atualiza a câmera conforme alteração. A cada pontuação o objeto principal fica maior

- * 2 : Adiciona cores aos objs
- * 4 : Adiciona cores aos obj principal
- * 6 : Troca para projeção perspectiva e câmera móvel
- * 8 : Utiliza modelo de iluminação de Lambert, gouraud shading
- * 10 : Adiciona texturas aos objs e ao obj principal
- * 12 : Começa a desenhar objs com maior complexidade de triângulos (cow.obj+)
- * 14 : Câmera segue o obj principal
- * 16 : Iluminação de Lambert, Phong shading
- * 18 : Câmera livre em primeira pessoa
- * 20 : Iluminação de Phong, gouraud shading
- * 22 : Iluminação de Phong, Phong shading
- * 24 : Iluminação relativa a fonte de luz (em vez de fonte de luz global única para cada obj)
- * 28 : Iluminação de Blinn-Phong, Phong shading

5. Desenha Frame:

- Desenha obj principal, plano e fila de objs do estado do jogo, utilizando view, câmera e shaders definidos anteriormente.

- Detecta colisões entre o objeto principal e o plano. Se o obj principal cruzar para fora do plano, é movido de volta para dentro.
 - Detecta colisões entre obj principal e objs da fila. Uma colisão entre obj principal e outro é caracterizado pela interseção de suas bounding box ou da interseção do ponto da câmera com a bounding box do obj.
 - Se uma colisão for detectada, remove objeto colidido da fila de objs, adiciona 1 na pontuação do usuário e altera `should_add_obj` para true.
 - Se o modo de curvas de Bézier estiver ligado, faz uma translação de todos os objs da fila utilizando a equação da curva e o parametro de tempo
6. Guarda tempo de iteração de renderização no timer e força framerate especificado pelo estado do jogo.
 7. Se `should_break` é verdadeiro, encerra programa. Se não, volta ao começo.

4 Funcionamento da Aplicação

4.1 Controles

A habilitação de um controle depende da pontuação do usuário e do estado do jogo. Por exemplo, não é possível mover a câmera até a câmera livre ser "desbloqueada".

- Personagem Principal
 - W,A,S,D: Controla movimentação
 - Mouse: controla ângulos da câmera livre e direção do movimento (quando habilitada)
- Qualquer Câmera:
 - O: Muda para projeção ortográfica
 - P: Muda para projeção perspectiva
- Câmera Look at:
 - Setas do teclado controlam posição da camera (quando habilitado, se existir camera look-at)
- Câmera Livre:
 - Mouse: controla ângulos da câmera livre e direção do movimento (quando habilitada)
- Iluminação ambiente:
 - 1, 2, 3: Diminui parâmetros x,y,z do vetor de espectro da iluminação ambiente

- 7, 8, 9: Aumenta parâmetros x,y,z do vetor de espectro da iluminação ambiente
- PageUp: Aumenta intensidade do espectro de iluminação ambiente
- PageDown: Diminui intensidade do espectro de iluminação ambiente
- Iluminação global:
 - F1, F2, F3 : Aumenta parâmetros x,y,z do vetor de direção da fonte de luz
 - F7, F8, F9 : Diminui parâmetros x,y,z do vetor de direção da fonte de luz
 - Numpad 1, 2, 3: Diminui parâmetros x,y,z do vetor de espectro da iluminação global
 - Numpad 7, 8, 9: Aumenta parâmetros x,y,z do vetor de espectro da iluminação global
 - Home: Aumenta intensidade do espectro de iluminação global
 - End: Diminui intensidade do espectro de iluminação global
- - Posição da Iluminação relativa (habilitada após 28 pontos):
 - * Setas do teclado: Posição x,z da fonte de luz relativa
 - * Insert/End: Posição y da fonte de luz relativa
- Outros:
 - R: Reinicia parametros de iluminação com os valores padrão
 - +: Adiciona um ponto e muda should_add_obj para true
 - -: Remove um ponto e muda should_add_obj para true
 - B: Liga movimentação dos objs em curvas de Bézier

4.2 Funcionamento

O funcionamento da aplicação é simples: O jogador deve utilizar seu personagem principal e coletar outros objetos espalhados pelo plano, colidindo com os mesmos. Ao colidir com um objeto, o mesmo some e o obj principal cresce. O jogo termina quando o usuário cresce a tal ponto que todos os objetos novos gerados nascem dentro do obj principal, encerrando qualquer possibilidade de continuar o jogo.

Objetos novos são gerados aleatoriamente, com propriedades de refletância especular, difusa, e tipos de textura, mapeamento de textura e modelo aleatórios. Alguns objetos são especiais (puramente especulares ou difusos ou realmente completamente aleatórios), e alguns tem um tipo de mapeamento de textura e propriedades mais padronizadas.

O jogador pode controlar o objeto principal utilizando o mouse e as teclas W,A,S,D (descritos na seção acima). A qualquer momento é possível pressionar

B e ativar o modo de Bézier, animando todos os objetos e move eles em uma curva (de Bézier).

As teclas + e - podem ser utilizadas para alterar a pontuação, se o processo de adquirir objetos para avançar o estado do jogo for muito tedioso.

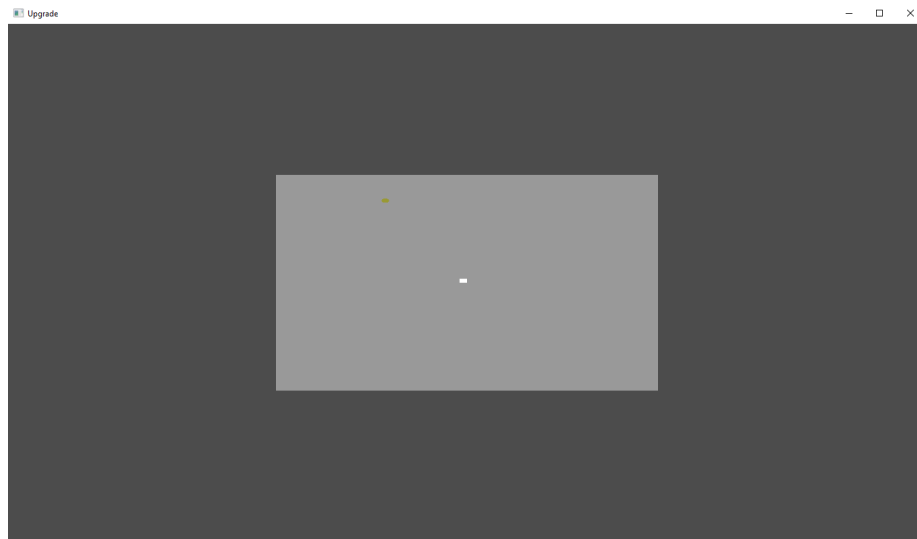


Figura 1: Início do jogo: Projeção ortográfica e câmera vertical simulam 2d

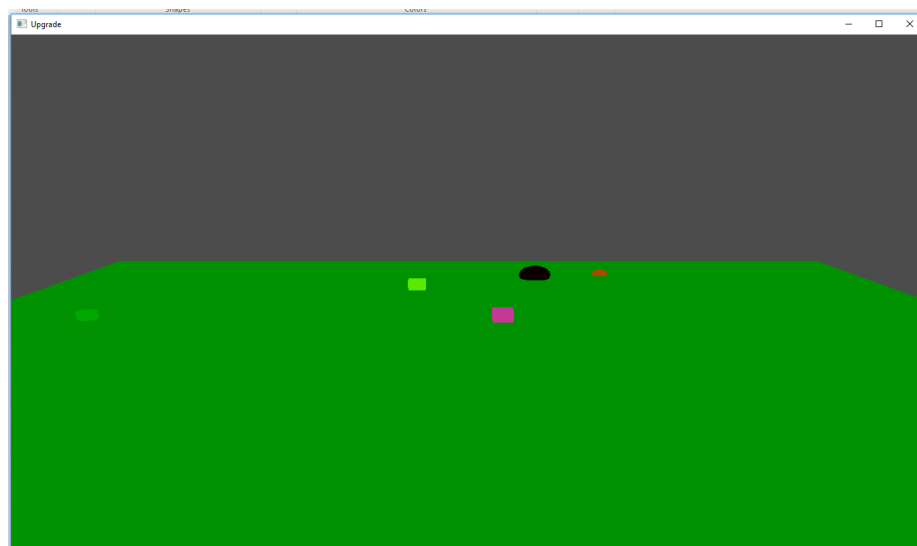


Figura 2: 6 pontos: Projeção perspectiva e cor

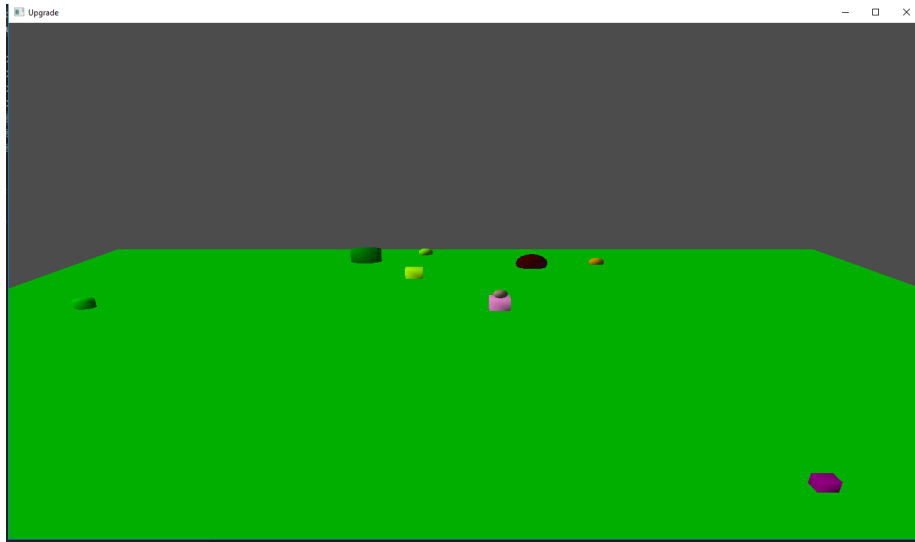


Figura 3: 8 pontos: Iluminação de lambert, gouraud shading

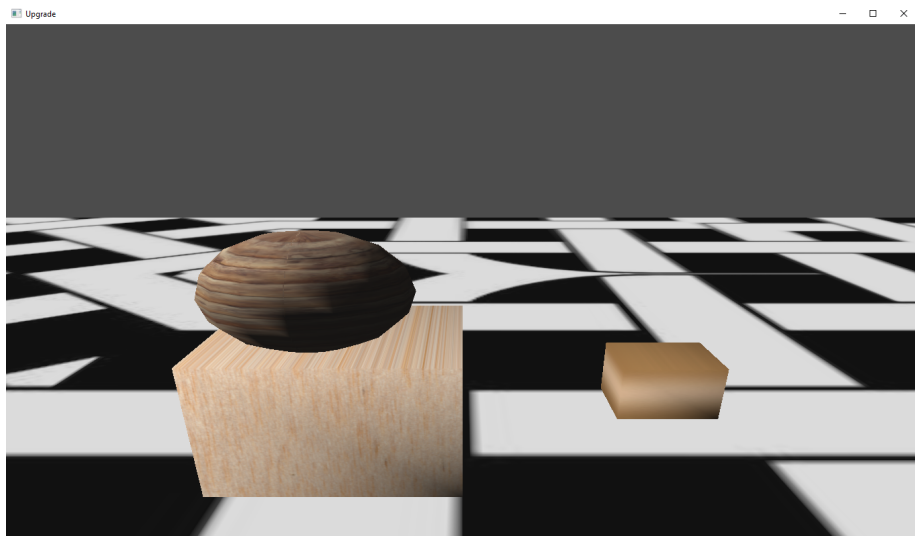


Figura 4: 14 pontos: Texturas

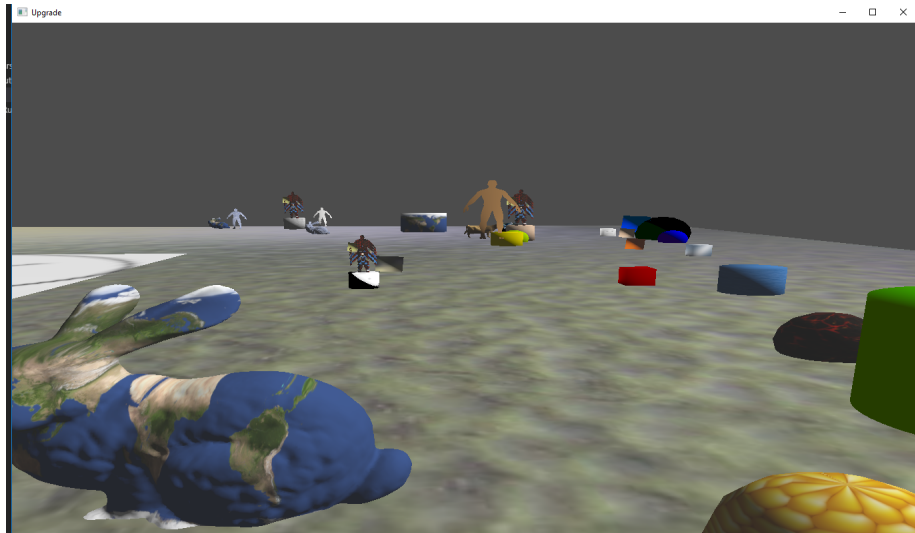


Figura 5: 28 pontos: Phong, phong shading



Figura 6: 28 pontos: Phong e fontes de luz relativas

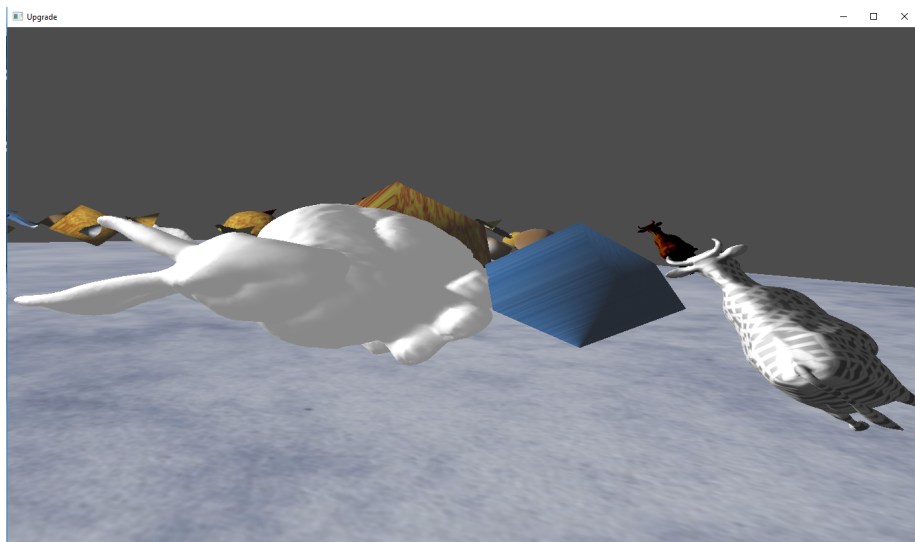


Figura 7: Final: Blinn-phong, phong shading, movendo em curva de Bézier

5 Requisitos

- Utilizar as matrizes que vimos em aula:
 - Arquivo `src/models/matrix.rs`
 - Utilizado para transformações em todos os objs da cena
- Interação com o usuário: Usuário controla cena e objeto principal com teclado e mouse
- Objetos virtuais representados através de malhas poligonais complexas
 - SceneObj carrega objs complexos. Vew `cow.obj`, `naked.dude.obj`
 - Objs são possíveis de composição e aplicam transformações hierárquicas
- Transformações geométricas de objetos virtuais.
 - Objeto principal é controlado e cresce conforme pontos são adquiridos. B liga curvas de Bézier
- Controle de câmeras virtuais
 - São implementadas duas cameras, uma look-at e uma livre.
- No mínimo um objeto virtual deve ser copiado com duas ou mais instâncias, isto é, utilizando duas ou mais Model matrix aplicadas ao mesmo conjunto de vértices
 - Todos os objetos são copiados a partir de instancias base, utilizando model matrix diferentes (além de outros)
- Testes de intersecção entre objetos virtuais.
 - É Implementado um teste `bounding_box` - `bounding_box` para colisão entre objs
 - Um teste `bounding_box` - ponto para colisão entre objs e câmera do obj principal
 - Um teste plano - `bounding_box` para detecção do obj principal nos limites do plano.
- Modelos de iluminação de objetos geométricos.

São implementados os seguintes modelos de iluminação e shading:

 - lambert e gouraud shading
 - phong e gouraud shading
 - lambert e phong shading
 - phong e phong shading
 - blinn-phong e phong shading

- Mapeamento de texturas

São mapeadas mais de 10 texturas aleatoriamente sobre objs

Mapeamento esférico, cilíndrico, linear xy, linear xz implementado.
- Curvas de Bézier.

Ao pressionar B todos os objetos começam a se mover em uma curva de Bézier e rotacionar aleatoriamente
- Animação de Movimento baseada no tempo.

Um framerate máximo é forçado, e as animações são parametrizadas pelo tempo de cada frame.

Referências

- 1 RUST. Disponível em: <https://www.rust-lang.org/>.
- 2 ARE-WE-GAME-YET? Disponível em: <http://arewgameyet.com/>.
- 3 GLUTIN. Disponível em: <https://github.com/rust-windowing/glutin>.
- 4 GL-RS. Disponível em: <https://github.com/brendanzab/gl-rs>.
- 5 TOBJ - Tiny OBJ Loader. Disponível em: <https://github.com/Twinklebear/tobj>.