Dariusz Maciejewski

– Camera Fighter –
Gra komputerowa

Promotor: prof. nzw. dr hab. Przemysław Rokita

Pracownia problemowa magisterska 2
- raport -

Opis zadania:

Camera Fighter ma być grą umożliwiającą przeprowadzanie wirtualnych pojedynków oraz wspierającą naukę sztuk walki. Zestaw trzech kamer będzie przechwytywał zdjęcia gracza. Po ich analizie, program rozpozna pozycję jaką przyjął gracz. W pozycji takiej zostanie pokazany wirtualny wojownik. Będzie to umożliwiało nowatorskie sterowanie postacią oraz w efekcie przeprowadzania wirtualnych walk z innymi graczami, bądź naukę poprawnych pozycji i ruchów w wybranych sztukach walki.
Moim zadaniem jest stworzenie logiki, fizyki oraz silnika 3D napędzającego całą grę. Przygotowany przeze mnie program zostanie następnie zintegrowany z przygotowywaną przez kolegę Michała Grędziaka, biblioteką odpowiedzialną za rozpoznawanie pozycji gracza.

Stan pracy na początku drugiego semestru:

Na początku drugiego semestru miałem gotowy szkielet uniwersalnej gry komputerowej. Pozwalał on na utworzenie pustej aplikacji OpenGL oraz wspierał podział kodu na niezależne od siebie logicznie sceny, takie jak menu, tryb rozgrywki, konsola czy konfiguracja.
Ponadto miałem przygotowany system sterowania kamerami z wykorzystaniem matematyki kwaternionów oraz własnego algorytmu obliczania obrotów.
Miałem również gotową klasę reprezentującą bazowy obiekt w grze, z którego obecnie wywodzą się różne typy modeli (3ds, 3dx).

Zakres prac wykonanych w drugim semestrze:

1.Klasa zarządzająca wejściem InputManager

Uniwersalny szkielet aplikacji został poszerzony o klasę singleton `InputManager`. Pośredniczy ona między zdarzeniami wejścia a aplikacją. Dzięki niej wszystkie sygnały wejścia mogą być dowolnie mapowane na wewnętrzne kody wykorzystywane przez logikę aplikacji. Użytkownik ma możliwość dowolnego skonfigurowania zachowania klawiatury w programie.

```
std::string Buffer;
void AppendBuffer(unsigned char charCode);
```

Metoda ta dodaje literę do bufora. Bufor jest wykorzystywany przez elementy programu wymagającego wprowadzania tekstu, np. przez konsolę.

```
void SetKeyState(int keyCode, bool state);
```
Ustawia stan sygnału wejściowego na podstawie mapowania klawisza na sygnał (wciśnięty lub nie).
```
void SetInputState(int inputCode, bool state);
bool GetInputState(int inputCode);
```
Ustawianie i zwracanie stan sygnału wejściowego.

```
void SetScene(char *scene);
```
Ustawia bieżącą scenę. Każda scena może mieć inne mapowania klawiszy.

```
int GetInputCode(int keyCode);
```
Zwraca kod sygnału na jaki jest mapowany klawisz.
```
void SetInputCode(int keyCode, int inputCode);
```
Ustawia mapowanie klawisza na kod sygnału.

```
void ClearMap();
```
Czyści wszystkie mapowania klawiszy.

```
void LoadMap(char *fileName);
void SaveMap(char *fileName);
```
Wczytywanie i zapisywanie mapowania na dysk.

```
std::string GetKeyName(int keyCode);
void LoadKeyCodeMap(char *fileName);
```
Pobieranie nazwy klawisza, wczytywanie tablicy mapującej klawisze na ich nazwy.

2.Bazujące na uchwytach menadżery tekstur i czcionek

Stworzyłem klasy TextureMgr oraz FontMgr. Są to klasy singleton, dzięki czemu z każdego podsystemu gry mamy szybki i przejrzysty dostęp do funkcjonalności jaką zapewniają.
Oba menadżery opakowują bazowy szablon menadżera zasobów, wykorzystującego technikę uchwytów do szybkiego i bezpiecznego odwoływanie się do zasobów.
Na poziomie menadżerów zaimplementowane jest zliczanie referencji. Aby zasób został usunięty z pamięci, musi zostać zwolniony tyle razy, ile razy był pobrany przez moduły zewnętrzne.
Menadżer tekstur potrafi wczytywać pliki typu BMP i TGA, natomiast menadżer czcionek pozwala na renderowanie dwu i trójwymiarowego tekstu.
Po przeładowaniu OpenGL (np. podczas zmiany tryby wyświetlania), uchwyty zachowują swoją ważność. Zasoby na które wskazywały zostają oznaczone przez menadżera jako błędne i zostaną ponownie załadowane przy pierwszej próbie odwołania się do nich poprzez uchwyt.

3.Obsługa modeli zapisanych w plikach 3ds oraz we własnym formacie plików

Dzięki darmowej bibliotece lib3ds, mam możliwość wczytywania plików w formacie 3ds. Jest to popularny formatu, do którego można eksportować modele z większości programów do obróbki modeli 3D. Modele 3ds konwertuję na potrzeby gry na własny, uproszczony format o rozszerzeniu 3dx. Format ten zawiera tylko niezbędne dla gry dane, oraz dodatkowo dla przyspieszenia wczytywania, wyliczone uprzednio wektory normalne.

Format 3dx umożliwia również zapisywanie modeli wzbogaconych o dane szkieletu, niezbędne do przeprowadzania animacji szkieletowej.

4.Usprawnienie kodu odpowiedzialnego za renderowanie

Kod odpowiedzialny za renderowanie modeli został zrewidowany i poprawiony. Zrezygnowałem z mało wydajnego trybu bezpośredniego (stosowanie funkcji `glVertex*`, `glNormal*`, itp.). Taki sposób dostarczania wierzchołków cechuje duży narzut czasowy funkcji. Dla sceny składającej się z kilkudziesięciu tysięcy wierzchołków należało wywołać kilkaset tysięcy funkcji. Obecnie korzystam krokowego przesyłania danych udostępnianego przez funkcje z rodziny `gl*Pointer` i `glDrawElements`. Zaowocowało to kilkukrotnym wzrostem wydajności renderowania. W przyszłym semestrze planuję dalszą poprawę efektywności renderowania, dzięki zastosowaniu rozszerzenia Vertex Buffer Objects (VBO).

5.OGL vertex i fragment shader

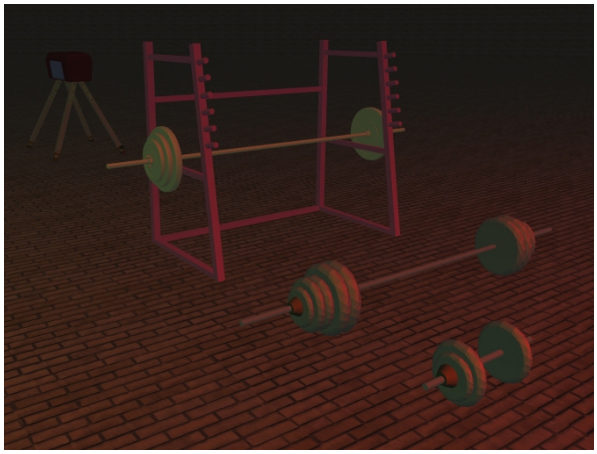
Napisałem programy OGL dla jednostki przetwarzającej wierzchołki oraz jednostki przetwarzającej powierzchnię modelu Programy karty graficznej zapewniają usprawnione oświetlenie (per fragment lighting). Jest ono ładniejsze i bardziej realistyczne, niż to zaimplementowane w standardowym kodzie karty graficznej. Jego wadą jest niestety negatywny wpływ na prędkość renderowania.
W przyszły semestrze planuję ponadto na zaimplementowanie animacji szkieletowej po stronie programu vertex shadera.
Niestety rozszerzenie vertex i fragment shadera jest dostępne jedynie na nowszych kartach graficznych, więc z ich zalet mogą korzystać tylko posiadacze nowszych komputerów.

6.Rozpoczęta praca nad animacją szkieletową modeli

W bieżącym semestrze rozpocząłem pracę nad implementowaniem animacji szkieletowej postaci. Zapoznałem się z teorią oraz przygotowałem struktury danych niezbędne do przechowywania informacji o szkielecie i powiązania go z modelem. Pozostaje mi zaimplementowanie algorytmu (po stronie vertex shadera oraz po stronie programu, dla starszych kart graficznych) oraz stworzenie możliwości definiowania szkieletów.

Podsumowanie semestru oraz plany na kolejny semestr:

W obecnym semestrze poszerzyłem swą wiedzę o OpenGL. Uniwersalny kod aplikacji jest już praktycznie skończony, podobnie statyczna część silnika graficznego.
W kolejnym semestrze planuję zbadać i zaimplementować możliwości udostępniane przez rozszerzenie karty graficznej Vertex Buffer Object. Ponadto zamierzam



dokończyć rozpoczętą pracę nad animacją szkieletową oraz oprogramować fizykę.
Zaimplementowanie animacji szkieletowej umożliwi rozpoczęcie integracji gry z biblioteką Video Motion Capture, przygotowywaną przez Michała Grędziaka, odpowiedzialną za rozpoznawanie pozycji przyjętej przez gracza.

Literatura:

Zagadnienia związane z programowaniem gier

'Perełki programowania gier – tom 1' pod redakcją Marca DeLoura, Wydawnictwo Helion 2002

OpenGL

The Red Book
http://fly.srk.fer.hr/~unreal/theredbook/

The Blue Book
http://www.rush3d.com/reference/opengl-bluebook-1.0/index.html

NeHe Productions
http://nehe.gamedev.net/

Jerome Lessons
http://jerome.jouvie.free.fr/OpenGl/Lessons.php

Jerome Tutorials
http://jerome.jouvie.free.fr/OpenGl/Tutorials1-5.php

Tulane.edu tutorial
http://www.eecs.tulane.edu/www/Terry/OpenGL/Introduction.html#Introduction

Dokumentacja rozszerzenia VBO
http://www.spec.org/gpc/opc.static/vbo_whitepaper.html

Dokumentacja rozszerzeń OpenGL
http://www.opengl.org/documentation/specs/man_pages/hardcopy/GL/html/

Animacja szkieletowa

Animacja szkieletowa z wykorzystaniem shaderów wierzchołków
http://grafika.iinf.polsl.gliwice.pl/doc/09-SKE.pdf

Podstawy animacji komputerowej
http://sound.eti.pg.gda.pl/student/sdio/12-Animacja.pdf