GKOM - Projekty

Wymagania projektu

W ramach projektu należy stworzyć program, który będzie realizował opisane w temacie funkcje.

Głównym językiem programowania może być język Python lub C++. Do realizacji funkcji graficznych należy wykorzystać bibliotekę OpenGL wraz z językiem shaderów GLSL.

Za projekt można uzyskać maksymalnie $x \times 15p.$, gdzie x to liczba osób w zespole. Ocenie w ramach projektu podlegają:

1. Działanie programu - realizacja funkcji (8 p.)
2. Efekty wizualne - prezentacja działania programu oraz kroku algorytmu w przyjemnie wizualny sposób (przygotowanie modeli, scenerii itd.) (2 p.)
3. Jakość kodu (3 p.)
4. Prezentacja wykonana na ostatnim wykładzie (2 p.)

Projekt musi być pokazany odpowiedniemu prowadzącemu przed terminem ostatniego wykładu. Dodatkowo, brak prezentacji na ostatnim wykładzie skutkuje niezaliczeniem projektu. Terminem ostatecznym oddania projektu jest 22.01.2024.

1 Model Viewer

Prowadzący: dr inż. Łukasz Dąbała

W ramach projektu należy stworzyć program, który będzie umożliwiał oglądanie modeli 3D oraz ich właściwości. W tym celu należy zaimplementować:

1. wsparcie dla różnego typu plików wejściowych z modelem 3D: obj, alembic
2. cieniowanie Phonga
3. interpretację właściwości materiału: diffuse, specular
4. wsparcie dla tekstur typu diffuse, specular
5. wsparcie dla świateł punktowych - możliwość dodawania nowych, edycja właściwości światła
6. możliwość wyświetlenia danych o modelu:
   (a) ilość wierzchołków, face'ów, krawędzi, współrzędnych uv
   (b) wizualizacja wektorów normalnych
   (c) wizualizacja modelu w postaci modelu szkieletowego (ang. *wireframe*)

   Wszystkie właściwości powinny być wyświetlone w tym samym oknie.
7. możliwość zobaczenia poszczególnych części obiektu tzn. jeżeli obiekt składa się z paru modelu możemy zobaczyć jego hierarchię oraz pokazać/ukryć część tej hierarchii.
8. kamerę perspektywiczną - możliwość poruszania się po scenie oraz obrotu
9. walidację modelu - brak konkretnych właściwości np. współrzędnych UV, powinien skutkować błędem

2 Mały program do animacji

Prowadzący: dr inż. Łukasz Dąbała

W ramach projektu należy stworzyć program, który będzie umożliwiał proste animacje dla wczytanych modeli.

W tym celu należy zaimplementować:

1. wsparcie dla podstawowego typu plików wejściowych z modelem 3D: obj
2. cieniowanie Phonga
3. interpretację właściwości materiału: diffuse, specular
4. wsparcie dla światła punktowego - możliwość poruszania światłem, edycja właściwości światła
5. możliwość zrobienia prostej animacji:
   (a) wstawianie i edycja klatek kluczowych dla transformacji (translacja, rotacja, skalowanie)
   (b) interpolacja stała i liniowa między klatkami
6. kamerę perspektywiczną - możliwość poruszania się po scenie oraz obrotu
7. możliwość renderu animacji do pliku wideo

3 Material Editing

Prowadzący: dr inż. Łukasz Dąbała

W ramach projektu należy stworzyć program, który będzie umożliwiał zaawansowaną edycję materiału dla wybranego pojedynczego obiektu. W tym celu należy zaimplementować:

1. wsparcie dla różnego typu plików wejściowych z modelem 3D: obj, alembic
2. cieniowanie Phonga
3. interpretację właściwości materiału: diffuse, specular
4. wsparcie dla tekstur typu diffuse, specular, normal
5. wsparcie dla świateł punktowych - możliwość dodawania nowych, edycja właściwości światła
6. możliwość nakładania efektu bump mappingu oraz displacement mappingu
7. możliwość zmian parametrów materiału (tzn. koloru czy wielkości nakładanego efektu)
8. kamerę perspektywiczną - możliwość poruszania się po scenie oraz obrotu

4 Subdivision

Prowadzący: dr inż. Łukasz Dąbała

W ramach projektu należy stworzyć program, który będzie umożliwiał wizualizację podziału obiektu:

1. wsparcie dla różnego typu plików wejściowych z modelem 3D: obj
2. cieniowanie Phonga
3. interpretację właściwości materiału: diffuse, specular
4. wsparcie dla tekstur typu diffuse, specular
5. wsparcie dla świateł punktowych - wystarczy pojedyncze światło
6. wsparcie dla algorytmów: Loop'a oraz Catmull-Clark służących do podziału siatki
7. możliwość wizualizacji iteracyjnej algorytmów wymienionych w poprzednim punkcie
8. kamerę perspektywiczną - możliwość poruszania się po scenie oraz obrotu

5 Renderer stereo

Prowadzący: dr inż. Łukasz Dąbała

W ramach projektu należy stworzyć program, który będzie umożliwiał rendering obrazów stereo. W tym celu należy zaimplementować:

1. wsparcie dla różnego typu plików wejściowych z modelem 3D: obj, alembic
2. cieniowanie Phonga
3. interpretację właściwości materiału: diffuse, specular
4. wsparcie dla tekstur typu diffuse, specular
5. wsparcie dla świateł punktowych - możliwość dodawania nowych, edycja właściwości światła
6. kamerę perspektywiczną - możliwość poruszania się po scenie oraz obrotu
7. kamerę stereo pracującą w różnych trybach i dającą możliwość sposobu trybu stereo:
   (a) tryb równoległy - kamery mają równoległe osie widoku
   (b) tryb zbieżny - osie widoku kamer przetną się. Miejsce przecięcia powinno być możliwe do ustawienia

   Kamera powinna umożliwiać ustawienie m.in. takich właściwości jak odległość oczu.
8. zapisanie wyników do plików graficznych
   (a) pojedyncze obrazy dla każdego z oczu
   (b) anaglif (przynajmniej red-cyan)
   (c) przelot (obraz przeznaczony dla monitorów z filtrem polaryzacyjnym)

6 Toon Shading

Prowadzący: dr inż. Łukasz Dąbała

W ramach projektu należy stworzyć program, który będzie umożliwiał kreskówkowe cieniowanie obiektu (ang. toon shading).

W tym celu należy zaimplementować:

1. wsparcie dla różnego typu plików wejściowych z modelem 3D: obj
2. interpretację właściwości materiału: diffuse, specular
3. wsparcie dla tekstur typu diffuse, specular
4. wsparcie dla świateł punktowych - wystarczy pojedyncze światło
5. kamerę perspektywiczną - możliwość poruszania się po scenie oraz obrotu
6. shadery niezbędne do przeprowadzenia cieniowania kreskówkowego:
   (a) powinno być możliwe remapowanie zakresów do konkretnych wartości (dzięki czemu obiekt będzie sprawiał wrażenie bardziej płaskiego)
   (b) krawędzie obiektu (zarówno zewnętrzne jak i wewnętrzne) powinny być podkreślone
   (c) należy dokonać stylizacji odbić światła

7 Billboarding

Prowadzący: dr inż. Łukasz Dąbała

W ramach projektu należy stworzyć program, który będzie umożliwiał wyświetlanie obiektów drzew oraz chmur z wykorzystaniem billboardów.

W tym celu należy zaimplementować:

1. wsparcie dla różnego typu plików wejściowych z modelem 3D: obj (chcemy, aby jakiś obiekt znajdował się w świecie)
2. interpretację właściwości materiału: diffuse, specular
3. wsparcie dla tekstur typu diffuse
4. cieniowanie Phonga
5. wsparcie dla świateł punktowych - wystarczy pojedyncze światło
6. kamerę perspektywiczną - możliwość poruszania się po scenie oraz obrotu
7. wsparcie dla obiektów typu drzewa oraz chmury, które do renderingu będą wykorzystywać metodę billboardingu
   (a) drzewa powinny być zrealizowane z wykorzystaniem billboardów typu axial
   (b) chmury powinny być zrealizowane z wykorzystaniem billboardów typu world-oriented
8. stworzenie skybox'a w celu wyświetlania otoczenia wokół kamery

8 Motion blur

Prowadzący: dr inż. Łukasz Dąbała

W ramach projektu należy stworzyć prostą grę wyścigową, w której dodany zostanie motion-blur

1. wsparcie dla różnego typu plików wejściowych z modelem 3D: obj. Interesuje nas głównie wczytanie trasy, natomiast dobrze będzie też w jakiś sposób zwizualizować wnętrze samochodu.
2. cieniowanie Phonga
3. interpretację właściwości materiału: diffuse, specular
4. wsparcie dla tekstur typu diffuse, specular
5. wsparcie dla świateł punktowych - wystarczy pojedyncze światło
6. kamerę perspektywiczną - przeczepioną do wnętrza samochodu
7. efekt motion-blur
   (a) obiekty powinny rozmazywać się wraz z ruchem
   (b) należy uwzględnić odległość od obiektów - obiekty bliższe rozmazują się bardziej niż dalsze

9 Shadow Mapping

Prowadzący: dr inż. Michał Chwesiuk

W ramach projektu należy stworzyć program, który będzie renderował prostą geometrycznie scenę, w której obiekty będą rzucać cienie.
W tym celu należy zaimplementować następujące punkty:

1. Implementację pięciu podstawowych brył (sześcian, walec, stożek, płaszczyzna itd.), bądź bardziej zaawansowanej geometrii (modele 3D).
2. Cieniowanie Phong'a dla wielu źródeł światła (minimum 3)
   (a) Rodzaje źródeł mogą być dowolne, kierunkowe, punktowe, lub stożkowe
   (b) Minimum 3 źródła światła
3. Wczytywanie sceny z pliku konfiguracyjnego, w którym zawarta jest lista obiektów oraz źródeł światła
   (a) Dla obiektów plik powinien zawierać rodzaj obiektu, pozycję, rotację, skalowanie i kolor
   (b) Dla źródła światła plik powinien zawierać rodzaj, pozycję, diffuse i specular
4. Implementacja algorytmu mapy cieni (Shadow Mapping)
   (a) Generowanie bufora ramki (framebuffer) dla każdego źródła światła
   (b) Rendering mapy cieni (opcjonalnie: podgląd mapy cieni w programie bądź zapis do pliku)
   (c) Aplikacja wygenerowanych mapy cieni w shaderze renderującym scenę
5. Przedstawienie eksperymentów z mapą cieni, zastosowanie różnych rozdzielczości, dodanie próbkowania Poissona, użycie *sampler2DShadow* itd.
6. Kamera perspektywiczna - możliwość poruszania się po scenie oraz obrotu

10 Deferred Shading

Prowadzący: dr inż. Michał Chwesiuk

W ramach projektu należy stworzyć program, który będzie renderował złożoną geometrycznie scenę z wieloma źródłami światła wykorzystując tzw. Forward Rendering.
W tym celu należy zaimplementować następujące punkty:

1. Przygotować scenę 3d składającą się z paru różnych figur geometrycznych
2. Zaimplementować algorytm Odroczonego Cieniowania (ang. Deffered Shading)
   (a) Implementacja render pass do generowania G-Buffera
   (b) Wykorzystanie G-bufferów w aplikowaniu cieniowania
3. Cieniowanie powinno obejmować składowe diffuse i specular
4. Cieniowanie powinno uwzględniać głębokość (bufor Z)
5. Cieniowanie powinno wspierać wiele źródeł światła
6. Przetestować scenę dla złożonej geometrycznie sceny (np. wiele obiektów o prostej geometrii)
7. Dodać wiele źródeł światła (Opcjonalnie: zaimplementować opcję dodawania N losowo wygenerowanych źródeł światła).
8. Kamerę perspektywiczną - możliwość poruszania się po scenie oraz obrotu

11 Environment Rendering

Prowadzący: dr inż. Michał Chwesiuk

W ramach projektu należy stworzyć program, który będzie renderował teren przy wykorzystaniu mapy wysokości.
W tym celu należy zaimplementować następujące punkty:

1. Przygotowanie geometrii środowiska opartą o wczytaną mapę głębokości
   (a) Wybór pliku zawierającego mapę głębokości może być zaimplementowana w kodzie, występować jako parametr programu, bądź nazwa pliku wczytywana jest z pliku konfiguracyjnego
2. Geometria powinna zawierać macierz punktów N x M, gdzie M i N to ilość próbek z mapy głębokości.
   (a) Punkty powinny być oddalone od siebie we współrzędnych X i Z o stałą odległość, najlepiej definiowaną przez zmienną.
   (b) Współrzędna Y pozycji wierzchołka powinna być wczytywana z mapy głębokości.
   (c) Kolejność renderowania wierzchołków powinna być przekazana do GPU jako Element Buffer Object.
3. Cieniowanie Phong'a
4. Kontekstowe teksturowanie
   (a) Przekazanie do shaderów mininum trzech tekstur.
   (b) Pobranie koloru dla wszystkich tekstur w fragment shaderze.
   (c) Wybranie, bądź blending tekstur w zależności od parametrów danego wierzchołka (np. pozycja, wysokość, nachylenie powierzchni itd.)
5. Dodanie do sceny obiektu imitującą wodę
   (a) Geometria wody jako quad rozciągający się na całą scenę.
   (b) Quad powinien mieć naniesioną teksturę wody.
   (c) Opcjonalnie: Teksturę wody można wykorzystać jako normal mapę, zaaplikowaną we fragment shaderze wody.
   (d) Dodać poruszanie się fal wody np. jako dodanie UV do tekstury modyfikowanej z czasem zmiennej uniform.

12 Particle system editor

Prowadzący: dr inż. Michał Chwesiuk

W ramach projektu należy stworzyć program, który będzie renderował system cząsteczek o ustalonych parametrach.
W tym celu należy zaimplementować następujące punkty:

1. Implementację systemu cząsteczek o ustalonych parametrach, z danym stanem początkowym i aktualizacją po czasie
2. System cząsteczek powinien zawierać następujące parametry:
   (a) Częstotliwość emitowania nowych cząsteczek
   (b) Czas życia danych cząsteczek
   (c) Zanikanie cząsteczek po zakończeniu ich czasu życia
   (d) Wielkość cząsteczek
   (e) Tekstura cząsteczek
   (f) Kierunek ruchu modyfikowany z czasem, z opcjonalnym uwzględnieniem grawitacji
   (g) Prędkość cząsteczek
   (h) Opcjonalnie: oddziaływanie cząstek na siebie
   (i) Opcjonalnie: Przezroczystość cząstek wraz z sortowaniem po kanale alfa
3. Wczytywanie parametrów systemu cząsteczek z pliku konfiguracyjnego
4. Możliwość modyfikowania parametrów systemy cząsteczek
5. Wczytywanie prostej sceny celem wizualizacji cząsteczek na zadanym tle
6. kamerę perspektywiczną - możliwość poruszania się po scenie oraz obrotu

13 Screen Space Ambient Occlusion

Prowadzący: dr inż. Michał Chwesiuk

W ramach projektu należy stworzyć program, który będzie renderował złożoną geometrycznie scenę z kilkoma źródłami światła i implementował technikę okluzji otoczenia.
W tym celu należy zaimplementować:

1. Wczytywanie sceny 3d składającej się z wielu różnorodnych otekstorowanych figur geometrycznych
2. Cieniowanie Phonga
3. Algorytm Screen space ambient occlusion
   (a) Implementacja algorytmu powinna wykorzystywać zasoby GPU, tj. być zaimplementowana w GLSL
   (b) Scena 3d powinna być przygotowana w taki sposób, aby możliwe było efektywne zaprezentowanie algorytmu
4. Przełączanie ilości sampli w hemisferze
5. Kamerę perspektywiczną - możliwość poruszania się po scenie oraz obrotu

14 Space Invaders

Prowadzący: dr inż. Michał Chwesiuk

W ramach projektu należy stworzyć program, który będzie odwzorowywał popularną grę komputerową Space Invaders, wykorzystując współczesne techniki renderingu.
W tym celu należy zaimplementować następujące punkty:

1. Przygotować scenę 3d składającą się z macierzy statków przeciwników oraz pojazdu gracza
2. Implementacja systemu sterowania statkiem gracza.
   (a) Poruszanie się w trójwymiarze
   (b) Strzelanie, czyli generacja obiektu - pocisków
   (c) Kolizja statku gracza z pociskami przeciwników
3. Implementacja systemu przeciwników
   (a) Poruszanie się przeciwników zgodnie z ustaloną logiką gry
   (b) Losowa generacja pocisków poruszających się w kierunku gracza
   (c) System kolizji statków przeciwnika z pociskiem gracza
4. Implementacja wybranych technik współczesnego renderingu:
   (a) Cieniowanie Phonga
   (b) Prosty system cząsteczek
5. Implementacja podstawowych elementów gry
   (a) Rozpoczęcie gry
   (b) Obsługa wygranej i przegranej
   (c) System punktacji
6. Implementacja dodatkowych elementów gry
   (a) Power-up'y - szybsze pociski / dodatkowe życie
   (b) Przeszkody na scenie

15 Dataset Generation

Prowadzący: dr inż. Michał Chwesiuk

W ramach projektu należy stworzyć program, który generował obrazy zdefiniowanej sceny z perspektywy wielu kamer, oraz dodatkowe obrazy zawierające etykiety zawierającej się na scenie obiektów.
W tym celu należy zaimplementować następujące punkty:

1. Implementację wczytywania obiektów 3D z jednego lub wielu szeroko stosowanych formatu pliku (obj, fbx, ply itp.) wraz z teksturami.
2. Możliwość definiowania obiektów na scenie wraz z ich pozycją, rotacją, itp. oraz unikalnym identyfikatorem w wybrany sposób:
   (a) Implementacja interface'u graficznego
   (b) Wczytywanie parametrów z pliku np. json.
3. Implementacja kamery perspektywicznej oraz możliwość definiowania wielu kamer, w podobny sposób co obiekty sceny. Każda kamera powinna posiadać:
   (a) Pozycje, kierunek patrzenia i wektor góry / macierz rotacji
   (b) Pole widzenia (field of view)
   (c) Czas przejścia (ilość klatek) do kolejnej kamery
4. Dwa tryby renderingu:
   (a) Klasyczny rendering obiektów sceny wraz z oświetleniem Phonga i wykorzystaniem tekstur.
   (b) Rendering płaskich kolorów zależnych od identyfikatora obiektu.
5. Procedurę generowania datasetu:
   (a) Tworzenie unikalnego katalogu dla zestawu danych
   (b) Ustawienie pierwszej kamery jako aktywnej
   (c) Generacja dwóch obrazów:
      i. Z obrazem sceny
      ii. Z kolorami wskazującymi identyfikatory obiektu
   (d) Ustawienie nowej pozycji kamery, której parametry są interpolowane między aktualną kamerą, a następną, zgodną z parametrem czasu przejścia.
   (e) Powtórzenie poprzednich punktów do momentu wyrenderowania obrazów dla ostatniej zdefiniowanej kamery.