Wydział Matematyki i Nauk Informacyjnych

Dokumentacja projektu
Grafika 3D – „Iluminacje świąteczne”

Jadwiga Słowik

Wersja 1.0
21 lutego 2017

Spis treści

1 Specyfikacja

1.1 Opis biznesowy

Niniejsza aplikacja zawiera trójwymiarową scenę 3D przedstawiającą fragment parku z choinkami w nocy. Ów park jest ozdobiony „iluminacjami" świątecznymi. Są to kolorowe lampki ułożone we wzory, oświetlające całą scenę. Na gruncie widoczne są rozbłyski owych lampek.

Na scenie znajduje się również obiekt (turkusowy kot), którym użytkownik może sterować w czterech kierunkach. Ponadto, użytkownik może patrzeć na scenę przy pomocy jednej z trzech kamer:

- Kamera nieruchoma, patrząca na całą scenę
- Kamera nieruchoma, śledząca kota
- Kamera ruchoma – patrząca „oczami" kota

oraz zapewniony jest wybór jednego z trzech dostępnych modeli cieniowania:

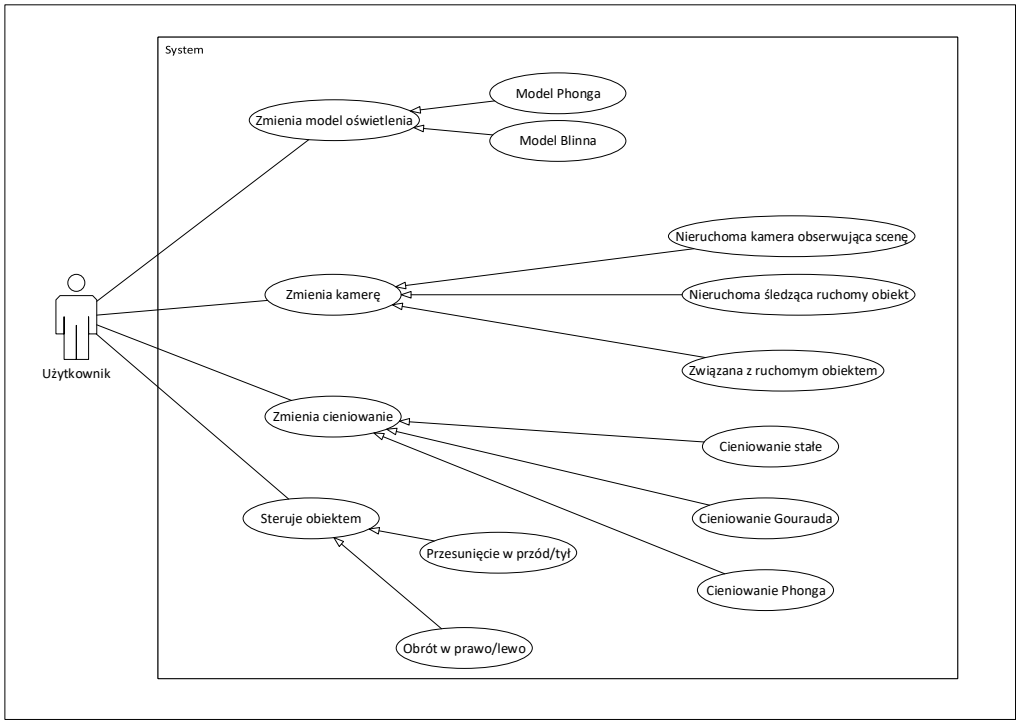
- Cieniowanie Gorauda
- Cieniowanie Phonga
- Cieniowanie płaskie

i wybór jednego z dwóch modeli oświetlenia:

- Oświetlenie Phonga
- Oświetlenie Blinna

1.2 Wymaganie funkcjonalne

Na poniższym diagramie przedstawiono zbiór przypadków użycia dla użytkowanika:



Rysunek 1: Przypadki użycia

Opisy przypadków użycia dla użytkownika:

Aktor	Nazwa	Opis	Odpowiedź systemu
Użytkownik	Zmienia model oświetlenia	Zmiania algorytm obliczania koloru światła dla pixela	Nieznaczna zmiana wyglądu rysowania sceny
	Zmienia kamerę na nieruchomą obserwującą scenę	Zmiana sposobu patrzenia na scenę	Zmiana obrazu widzianego przez użytkownika na obraz całej sceny
	Zmienia kamerę na nieruchomomą śledzącą ruchomy obiekt	Zmiana sposobu patrzenia na scenę	Zmiana obrazu widzianego przez użytkownika. Wraz ze zmianą położenia kota, ulega również zmianie obraz widziany na ekranie, ale kamera nie zmienia położenia.
	Zmienia kamerę na ruchomą związaną z ruchomym obiektem	Zmiana sposobu patrzenia na scenę	Zmiana obrazu widzianego przez użytkownika. Na ekranie przedstawiony jest obraz widziany „oczami" obiektu ruchomego. Ów obraz zmienia się po zmianie położenia obiektu ruchomego
	Zmienia cieniowanie	Zmienia sposób cieniowania na jeden z trzech możliwych (cieniowanie Phonga, Goraud a, płaskie)	Zmiana wyglądu sceny
	Steruje obiektem	Steruje obiektem (w jednym z trzech kierunków)	Zmiana położenia / ustawienia ruchomego obiektu

1.3 Harmonogram projektu

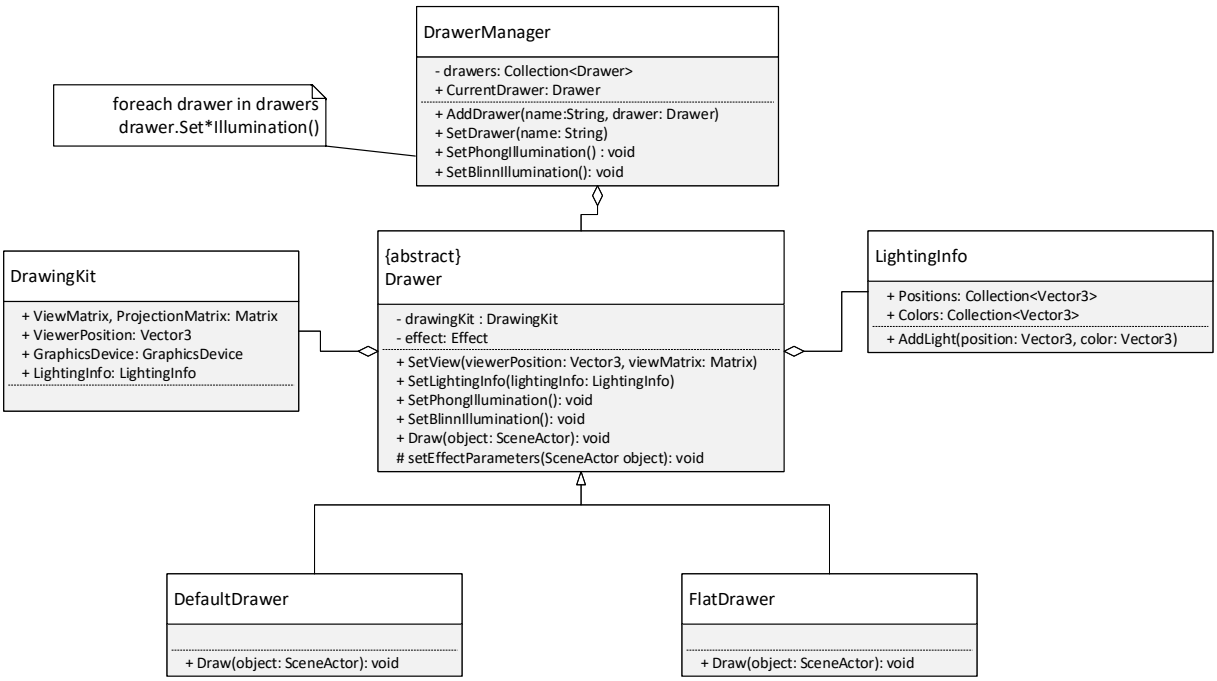
Nr	Zadanie	Data rozpoczęcia	Data zakończenia	Przeznaczony czas
1	Wstępna analiza zadania	5.12.2016	18.12.2016	12 dni
2	Przygotowanie specyfikacji	18.12.2016	28.12.2016	1 dzień
3	Wdrożenie technologii	28.12.2016	28.01.2017	31 dni
4	Projektowanie rozwiązania	28.01.2017	06.02.2017	9 dni
5	Implementacja	06.02.2017	20.02.2017	14 dni
6	Przygotowanie końcowej dokumentacji	20.02.2017	22.02.2017	2 dni
7	Oddanie projektu	22.02.2017	22.02.2017	1 dzień

1.4 Architektura rozwiązania

Rozwiązanie zostało podzielone na poszczególne częsci:

1.4.1 Rysowanie

Do rysowania przeznaczone są klasy pochodne klasy *Drawer*. Implementują one konkretny algorytm, korzystając z publicznych pól obiektu, który ma być rysowany (*SceneActor*).

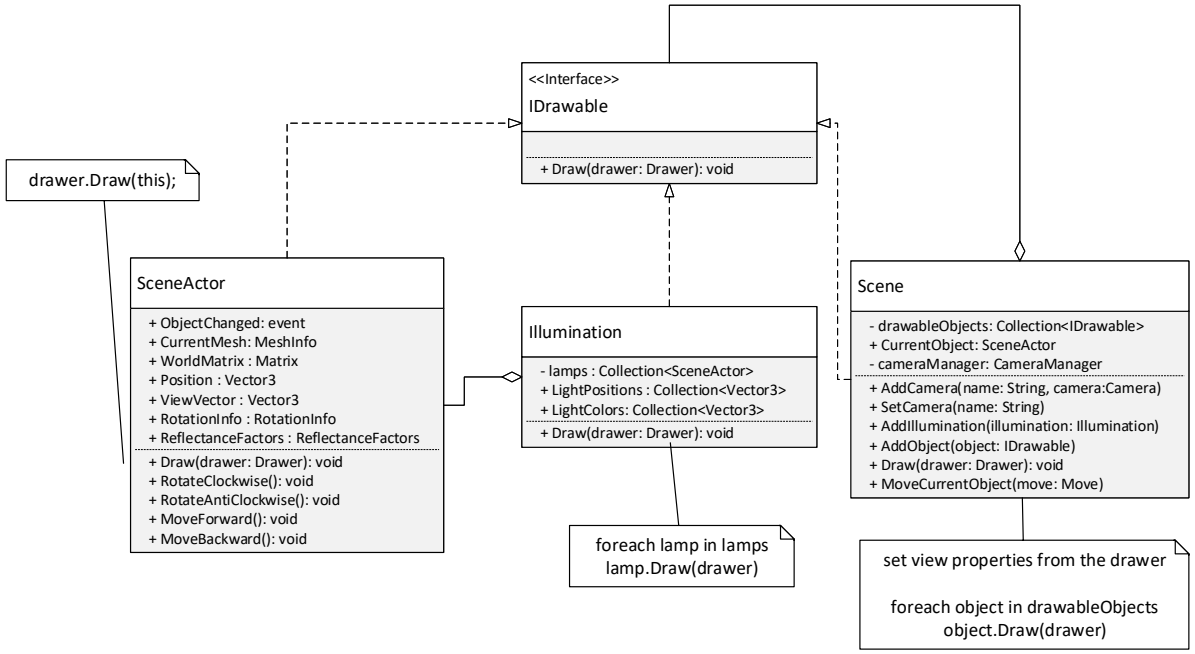


Rysunek 2: Narzędzia do rysowania

Każdy „rysownik” posiada „przybornik do rysowania” (*DrawingKit*) oraz „efekt”, w którym znajduje się implementacja shadera. Przed rozpoczęciem rysowania, w wymaganych polach shadera są ustawiane konkretne wartości odczytane z aktualnie przekazanego obiektu.

„Przybornik do rysowania” zawiera dane potrzebne do wykonania procedury rysowania, które nie są związane z rysowanym obiektem takie jak: dane związane z widokiem, referencja do sterownika karty graficznej, macierz Projekcji oraz dane związane z aktualnym oświetleniem.

Istniejącymi „rysownikami” zarządza *DrawerManager*, który zawiera do nich referencje i wie, który jest aktualnie ustawiony, tj. za pomocą którego wykonywać procedurę rysowania. Ponadto, interfejs „zarządcy” zapewnia nam możliwość dodawania nowych „rysowników” oraz zmiany aktualnego sposobu rysowania.



Rysunek 3: Obiekty rysowane na scenie

Rysowane mogą być obiekty implementujące interfejs *IDrawable* takie jak: *SceneActor*, *Illumination*, *Scene*.

Klasa *SceneActor* reprezentuje elementarny obiekt, który może być rysowany. Zawarte są w niej wszystkie cechy i właściwości, związane z jego wyglądem takie jak: aktualna siatka trójkątów (wewnątrz obiektu klasy *SceneActor* może być wiele siatek trójkątów. Dzięki temu, możemy symulować animację obiektu), informacje związane z położeniem, współczynniki pochłaniania/odbicia światła, „wektor patrzenia" (za jego pomocą wiemy, jak interpretować ruch obiektu do przodu/tyłu; ów wektor jest modyfikowany w momencie wykonywania obrotu obiektu).

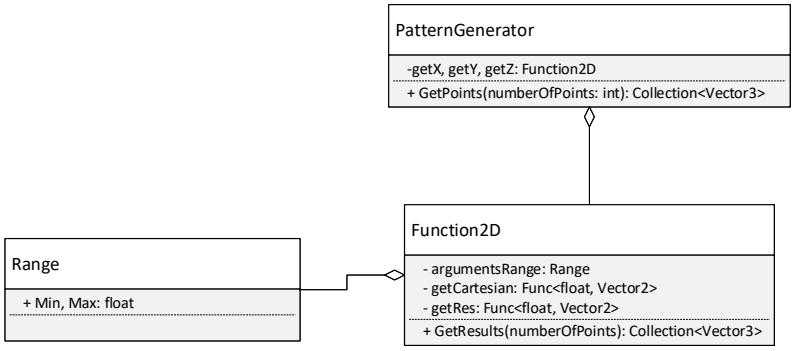
Klasa *Illumination* reprezentuje wzór światełek. Składa się ona z mniejszych obiektów reprezentujących pojedyncze elemenenty (obiekty klasy *SceneActor*). W celu optymalizacji wydajności aplikacji, palą się tylko niektóre światełka. Jednakże, nie jest to zauważalne przez obserwatora, gdyż świecące lampki oświetlają pozostałe i można odnieść wrażenie, że wszystkie światełka są zapalone.

Klasa *Scene* enkapsuluje pojedynczą scenę. Zapisane są w niej referencje do wszystkich obiektów, które się na niej znajdują. W szczególności, posiada referencję do obiektu, którym użytkownik porusza (*CurrentObject*). Co więcej, w scenie jest zapisana referencja do „zarządcy kamer", który jest odpowiedzialny za to, co użytkownik widzi na ekranie. Zarządzanie kamerami zostanie wyjaśnione w późniejszym podrozdziale.

Rysowanie zostało zaimplementowane przy pomocy wzorca projektowego „kompozyt" (obiekty rysowalne są budowane za pomocą mniejszych obiektów rysowalnych) i „strategia" (*DrawersManager*) oraz „podwójnego rozsyłania" (*ang. double dispatch*).

1.4.2 Tworzenie wzorów iluminacji

Każda z iluminacji składa się z wielu elementów (pojedynczych egzemplarzy wybranego modelu), które są ułożone przy pomocy danego wzoru.

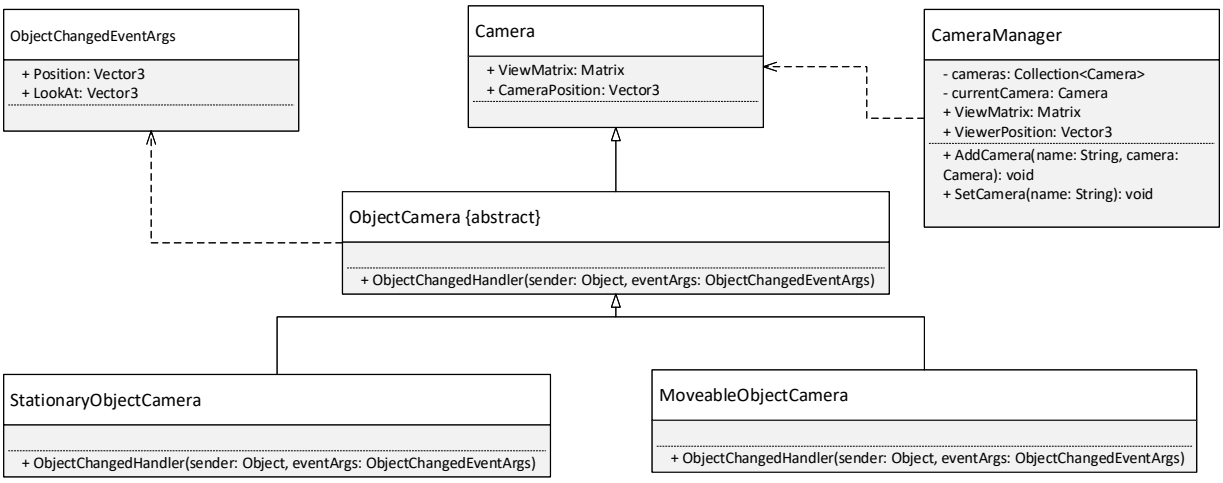


Rysunek 4: Komponenty użyte do generowania wzorów

Współrzędne obiektów są generowane przy pomocy klasy *PatternGenerator*. Aby otrzymać współrzędne punktów, w których mają się znajdować elementarne składowe danej iluminacji, musimy podać trzy funkcje dwuwymiarowe, które określają współrzędne x, y i z odpowiednio. Funkcje mogą być podane w dowolnej formie (przy pomocy współrzędnych kartezjańskich, polarnych itd), jednakże, dla każdej funkcji należy podać funkcję przejścia do współrzędnych kartezjańskich.

1.4.3 Zarządzanie kamerami

Zarządzanie kamerami odbywa się w podobny sposób do zarządzania „rysownikami". Mianowicie, za zarządzanie kamerami odpowiedzialny jest *CameraManager*, która posiada referencje do wszystkich aktualnie istniejących kamer i pamięta obecnie ustawioną kamerę. Ponadto, posiada dwie właściwości – *ViewMatrix* i *CameraPosition* odczytywane przy pomocy odpowiadających publicznych pól aktualnie ustawionej kamery.



Rysunek 5: Kamery

W aplikacji występują 3 rodzaje kamer – jedna „stała" i dwie związane z ruchomym obiektem. Owym rodzajom odpowiadają odpowiednio klasy *Camera* i dwie klasy pochodne abstrakcyjnej klasy *ObjectCamera*.

Jako że *ObjectCamera* reprezentuje klasę kamer związanych z ruchomym obiektem, to konieczne jest powiązanie danej kamery z konkretnym obiektem. W aplikacji realizowane jest to za

pomocą architektury zdarzeń (może to być również zaimplementowane za pomocą wzorca projektowego „Obserwator", jeśli interesujący nas język programowania nie udostępnia interfejsu zdarzeń). Mianowicie, w trakcie tworzenia obiektu kamery (związanej z obiektem), klasa kamery „rejestruje" się na otrzymywanie informacji o zajściu zdarzenia (zdarzenie *ObjectChanged*). Zatem obiekt „obserwowany" nie wie przez kogo jest obserwowany. Owe zdarzenie jest generowane po zmianie położenia danego obiektu.

Przed rozpoczęciem procedury rysowania ustawiane są aktualne dane dotyczące widoku (*ViewMatrix* i *CameraPosition*) zapisane w *CameraManager*.

2 Dokumentacja końcowa (powykonawcza)

2.1 Biblioteki

Aplikacja została zaimplementowana w technologii .NET z pomocą biblioteki graficznej *Monogame*.

2.2 Instrukcja użycia

2.2.1 Sterowanie obiektem

Sterowanie odbywa się przy pomocy strzałek:

- Obrót za pomocą strzałek „prawo" / „lewo"
- Ruch w przód/tył za pomocą strzałek „góra" / „dół"

2.2.2 Zmiana modelu oświetlenia

Dostępne są dwa modele oświetlenia:

- Ustawienie modelu Phonga za pomocą kombinacji klawiszy: `Alt + P`
- Ustawienie modelu Blinna za pomocą kombinacji klawiszy: `Alt + B`

2.2.3 Zmiana modelu cieniowania

Dostępne są trzy modele cieniowania

- Ustawienie modelu Phonga za pomocą kombinacji klawiszy: `Shift + P`
- Ustawienie modelu Gorauda za pomocą kombinacji klawiszy: `Shift + G`
- Ustawienie cieniowania płaskiego za pomocą kombinacji klawiszy: `Shift + F`

2.2.4 Zmiana kamery

Dostępne są trzy kamery:

- Zmiana na kamerę stałą za pomocą kombinacji klawiszy: `Ctrl + C`
- Zmiana na kamerę nieruchomą śledzącą ruchomy obiekt: `Ctrl + S`
- Zmiana na kamerę ruchomą, patrzącą „oczami" ruchomego obiektu: `Ctrl + M`