coffeecam

Bartosz Pieńkowski, Barnaba Turek

24 października 2011

Streszczenie

Celem serii projektów z przedmiotu Grafika Komputerowa jest utworzenie prostego silnika graficznego (i zapoznanie się od strony graficznej z podstawowymi zadaniami grafiki komputerowej, takimi jak rzutowanie, eliminacja elementów zasłoniętych i cieniowanie).

Zdecydowaliśmy się zaimplementować projekt w języku coffeescript.

Pierwsza część projektu polega na utworzeniu wirtualnej kamery. Kamera ma obsługiwać zmianę pozycji, ogniskowej i kąta, w którym jest skierowana.

1 Wirtualna kamera

1.1 Struktura projektu

Projekt podzielony jest na pliki .coffee, z których każdy zostaje skomplikowany do pliku .js (javascript). Zawartość każdego pliku wynikowego jest opakowana w anonimową funkcję, aby zniwelować ryzyko konfliktu nazw z innymi skryptami.

Zmienne i funkcje potrzebne do komunikacji pomiędzy modułami dostępne są za pomocą zmiennej coffecam.

Nie wszystkie obiekty przedstawione na diagramie 1 są w istocie klasami - z wyjątkiem klas **Camera** i **Controller** nie potrzebowaliśmy przechowywać żadnego stanu, a tylko logicznie podzielić udostępniane funkcje.

Plik *site* jest swojego rodzaju funkcją main, która uruchamia się po załadowaniu dokumentu przez przeglądarkę.

1.2 Działanie kamery

Obiekt kamery przechowuje macierz transformacji, początkowo ustawioną na macierz jednostkową (z dokładnością do znaku, ze względu na skrętności układów)[1]. Macierz transformacji obliczana jest na nowo po każdym ruchu kamery. Obliczenie nowej macierzy polega na pomnożeniu aktualnej macierzy przez żądaną transformacje.

Dostępne transformacje to przesunięcia i obroty kamery (Transformacje utworzylismy na podstawie wykładu[2]).

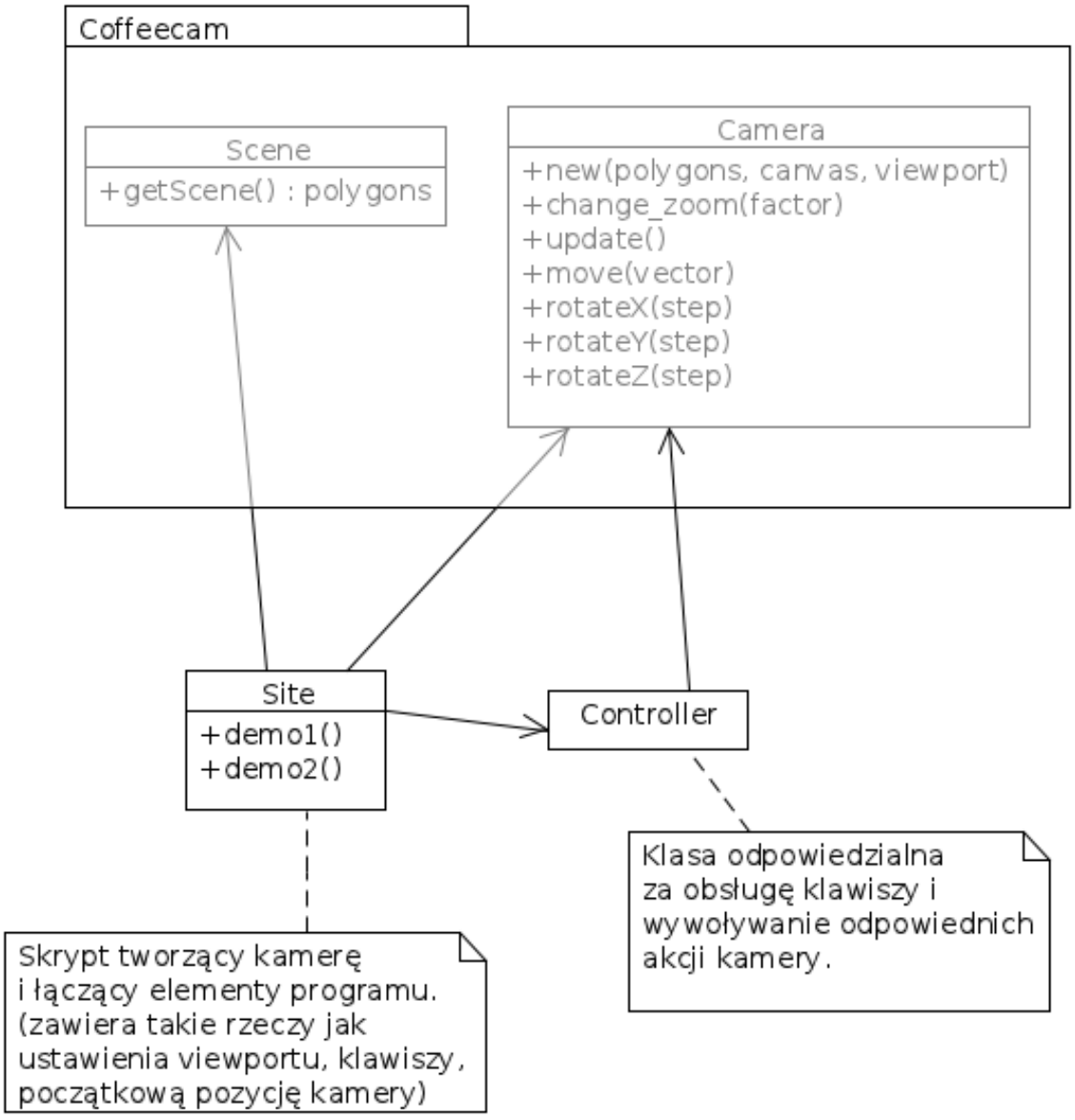
Po transformacji macierz transformacji mnożona jest przez macierz rzutowania (Obliczaną kiedy kamera jest tworzona i przy zmianie ogniskowej)[1]. Następnie każdy z punktów sceny mnożony jest przez macierz wynikową. Przygotowanie każdej nowej klatki realizuje metoda *update()* kamery.

1.3 Scena

Obiekty, które kamera wyświetla to wielokąty. Kamera zakłada, że punkty obiektu podane są w sposób uporządkowany.

Scena tworzona jest z czworokątów, ze względu na to, że rysujemy głównie prostopadłościany.

Rysunek 1: Struktura projektu



1.4 Pozostałe informacje o projekcie

Statyczne części projektu (strony, style) są tworzone za pomocą frameworku staticmatic, na podstawie źródeł (W językach HAML i SASS).

W projekcie wykorzystaliśmy javascriptową bibliotekę Sylvester wspierającą operacje na macierzach i wektorach.

Kod źródłowy projektu (wraz z aktualną dokumentacją) dostępny jest na serwisie Github (głównie katalog /src/coffee).

Projekt jest dostępny do testowania na następujących serwerach:

- Serwer 1[1]
- Volt[2] (Niekoniecznie aktualna wersja)

2 Eliminacja powierzchni zasłoniętych

Celem drugiego etapu projektu było zaimplementowanie wybranego algorytmu eliminacji powierzchni zasłoniętych.

Ze względu na prostotę sceny zdecydowaliśmy na algorytm malarski. Działanie tego algorytmu polega na rysowaniu obiektów w kolejności ustalonej przez ich odległość od kamery.

2.1 Implementacja

Algorytm nie zajął zbyt wiele miejsca i nie uznaliśmy za stosowne dodawać kolejnej klasy do obiektu. Dodaliśmy kod do klasy *Camera*.

Działanie algorytmu składa się z trzech kroków. W pierwszym kroku obliczamy współrzędne kamery w układzie odniesienia sceny.

Ponieważ kamera zawsze znajduje się w punkcie

$$P_{cam} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

i w każdej klatce zna swoją pozycję względem początku układu współrzędnych sceny (daną macierzą transformacji `@transformation`), możemy łatwo obliczyć położenie kamery w układzie współrzędnych sceny.

Transformacja, której szukamy (transformująca położenie z układu kamery do układu sceny) jest transformacją odwrotną do `@transformation`, zatem współrzędne kamery w układzie współrzędnych możemy policzyć w następujący sposób:

$$\mathtt{P}'_{\mathtt{cam}} = \mathtt{@transformation}^{-1} \times \mathtt{P}_{\mathtt{cam}}$$

Następnym krokiem jest obliczenie odległości środków ciężkości rysowanych wielokątów od punktu P'_{cam}. W tym celu obliczamy środek ciężkości wielokąta `barycenter`, gdzie P_k to kolejne wierzchołki wielokąta:‘

$$\mathtt{barycenter} = \frac{1}{\mathtt{polygon.length}} \sum_{\mathtt{k=0}}^{\mathtt{polygon.length}-1} \mathtt{P}_{\mathtt{k}}$$

i jego odległość od P'_{cam}:

$$\mathtt{distance} = |\mathtt{barycenter} - \mathtt{P}'_{\mathtt{cam}}|$$

Tak obliczone odległości przypisujemy do wielokątów, które następnie sortujemy wg. tych odległości (malejąco).

[1] `http://barnex.mooo.com`
[2] `http://volt.iem.pw.edu.pl/~pienkowb/`

Literatura

[1] Song Ho Ahn, OpenGL Projection Matrix, http://www.songho.ca/opengl/gl_projectionmatrix.html

[2] Dariusz Sawicki, Grafika komputerowa i wizualizacja, http://wazniak.mimuw.edu.pl/index.php?title=Grafika_komputerowa_i_wizualizacja