Projekt: rendering obrazów - dokumentacja

Michał Jaworski

21 stycznia 2018

1 Kompilacja programu

Program jest kompilowany z użyciem środowiska Cabal. Wymaga on zainstalowania następujących dodatkowych pakietów:

- *parsec*, używany do parsowania plików z opisem sceny
- *bmp*, używany do zapisywania obrazów w formacie BMP
- *gloss*, używany do wyświetlania obrazów

2 Używanie programu

Program należy wywołać z wiersza poleceń podając jako argument plik zawierający opis sceny. Po uruchomieniu, jeśli plik został prawidłowo wczytany, program wyświetli wygenerowany obraz oraz zapisze go w formacie BMP.

3 Format opisu sceny

Opis sceny wczytywany jest z pliku tekstowego w opisanym niżej formacie.

- Wiersze rozpoczynające się od znaku # są traktowane jako komentarze i ignorowane przez parser.
- Wielkość liter w słowach kluczowych nie ma znaczenia
- Wektory oraz punkty w przestrzeni trójwymiarowej przedstawione są jako trzy liczby rzeczywiste oddzielone białymi znakami
- W podobny sposób reprezentowane są kolory w postaci RGB, kolejne liczby oznaczają wartości na odpowiednich kanałach
- W parserze zdefiniowano również stałe reprezentujące podstawowe kolory. Są to: *black*, *white*, *red*, *green*, *blue*, *cyan*, *magenta* oraz *yellow*

Plik tekstowy opisujący scenę składa się z następujących części:

1. **Nagłówek** (obowiązkowy), każda z poniższych informacji poprzedzona jest odpowiednim słowem kluczowym:

- *imWidth* oraz *imHeight*: liczby całkowite reprezentujące odpowiednio szerokość oraz wysokość obrazka w pikselach, podanych jako liczby całkowite
- *canvWidth* oraz *canvHeight*: liczby rzeczywiste reprezentujące odpowiednio szerokość oraz wysokość prostokąta, przez który obserwowana jest scena
- *depth*: liczba rzeczywista reprezentująca odległość ogniska od płótna
- *bgColor*: opcjonalny parametr oznaczający kolor tła. Domyślnie jest to kolor czarny
- *rayDepth*: opcjonalny parametr określający maksymalną głębokość rekursji podczas śledzenia promieni. Domyślną wartością jest 4

2. **Źródła światła** (opcjonalne). Lista źródeł światła rozpoczynająca się słowem kluczowym *lights*. Dostępne są następujące źródła światła:

   - *directional* i c d: kierunkowe źródło światła o intensywności wyznaczonej przez liczbę rzeczywistą i, o kolorze c, świecące w kierunku wskazywanym przez wektor d
   - *spherical* i c x: punktowe źródło światła o intensywności i, kolorze c, znajdujące się w punkcie x

3. **Obiekty** (opcjonalne). Lista znajdujących się na scenie obiektów, rozpoczynająca się słowem kluczowym *objects*. Każdy obiekt przedstawiony jest w postaci *kształt powierzchnia*. Program udostępnia następujące kształty:

   - *sphere* x r: kula o środku x i promieniu r
   - *plane* x d: płaszczyzna zawierająca punkt x prostopadła do wektora d

   oraz następujące rodzaje powierzchni:

   - *diffusive* c: powierzchnia rozpraszająca światło, w kolorze c
   - *reflective*: powierzchnia odbijająca światło
   - *luminous* c: powierzchnia świecąca własnym światłem, w kolorze c
   - *mixed* t_1 s_1 $[t_2]$ $[s_2]$ ...: powierzchnia mieszana, gdzie wartości t_i wyznaczają proporcje, a s_i to mieszane rodzaje powierzchni

4 Dokumentacja kodu

4.1 Datatypes.hs

Plik zawiera definicje podstawowych typów danych wykorzystywanych w programie

- *Vector* - typ danych reprezentujący wektory w przestrzeni trójwymiarowej.
  Konstruktor:
  - *Vector Double Double Double* - wektor zadany poprzez współrzędne
- *(+.) :: Vector -> Vector -> Vector* - operator dodawania wektorów
- *(-.) :: Vector -> Vector -> Vector* - operator odejmowania wektorów
- *times :: Double -> Vector -> Vector* - mnożenie wektora przez skalar
- *dot :: Vector -> Vector -> Double* - iloczyn skalarny wektorów
- *sqVecLen :: Vector -> Double* - kwadrat długości danego wektora
- *vecLen :: Vector -> Double* - długość danego wektora
- *normalize :: Vector -> Vector* - normalizuje wektor, tzn. zwraca wektor o długości 1 równoległy do danego
- *Color t* - klasa reprezentacji kolorów
  - *cAdd :: t -> t -> t* - suma kolorów
  - *cTimes :: Double -> t -> t* - mnożenie koloru przez skalar
  - *cMult :: t -> t -> t* - iloczyn kolorów
  - *toWordList :: t -> [Word8]* - zamiana koloru na tablicę 4 wartości typu *Word8*, reprezentujących kolor w formacie RGBA32
  - *black :: t, white :: t* - stałe reprezentujące odpowiednio czerń i biel
  - Instancje: *Double*, *Vector*
- *Greyscale* - alias typu *Double*, reprezentacja odcieni szarości
- *RGB* - alias typu *Vector*, reprezentacja kolorów w postaci RGB
- *makeRGB :: Double -> Double -> Double -> RGB* - alias konstruktora *Vector*
- *red :: RGB, green :: RGB, blue :: RGB, cyan :: RGB, magenta :: RGB, yellow :: RGB* - stałe reprezentujące odpowiednio czerwień, zieleń, błękit, cyjan, fuksję i żółć

4.2 Geometry.hs

Plik zawiera typy danych do reprezentacji geometrii obiektów w przestrzeni trójwymiarowej oraz funkcje umożliwiające m. in. wyznaczanie punktów przecięcia promieni z obiektem czy promieni odbitych

- *eps :: Double* - stała reprezentująca odległość, o którą zostanie przemieszczony początek promieni odbitych oraz promieni wyznaczających cień
- *Ray* - typ danych reprezentujący promienie.
  Konstruktor:
  - *Ray { origin :: Vector, dir :: Vector }* - *Ray x d* tworzy promień wyznaczony przez półprostą o równaniu $y = x + td$ $(t > 0)$. Wektor d musi być wektorem znormalizowanym.
- *makeRay :: Vector -> Vector -> Ray* - normalizuje drugi argument, po czym wywołuje konstruktor *Ray*
- *getRayPoint :: Ray -> Double -> Vector* - *getRayPoint r t* zwraca punkt otrzymany po podstawieniu wartości t do równania opisującego promień r
- *reflectRay :: Vector -> Vector -> Ray -> Ray* - *reflectRay x n r* zwraca promień o początku w punkcie x (leżącym na promieniu r), powstały poprzez odbicie promienia r względem wektora normalnego n
- *Geometry* - typ danych reprezentujący geometrie obiektów w przestrzeni trójwymiarowej.
  Konstruktory:
  - *Sphere Vector Double* - kula opisana przez jej środek oraz promień
  - *Plane Vector Vector* - płaszczyzna opisana przez jeden z jej punktów oraz wektor normalny
- *makeSphere :: Vector -> Double -> Geometry* - alias konstruktora *Sphere*
- *makePlane :: Vector -> Vector -> Geometry* - normalizuje drugi argument, po czym wywołuje konstruktor *Plane*
- *normalVector :: Geometry -> Vector -> Vector* - wektor normalny prostopadły do powierzchni w danym punkcie
- *intersect :: Ray -> Geometry -> [Double]* - zwraca listę wartości t, dla których dany promień, opisany równaniem $y = x + td$, przecina dany obiekt. Współrzędne tych punktów można wyznaczyć za pomocą funkcji *getRayPoint*.

- *reflect :: Geometry -> Vector -> Ray -> Ray* - *reflect g x r* zwraca promień powstały po odbiciu promienia r w punkcie x należącym do obiektu g

- *LightSource t* - typ danych reprezentujący źródło światła, wykorzystujący reprezentację koloru t.
  Konstruktory:

  - *Directional t Vector* - światło kierunkowe o określonym kolorze padające w kierunku zadanym poprzez wektor normalny
  - *Spherical t Vector* - punktowe źródło światła zdefiniowane poprzez jego kolor i położenie

- *makeDirectional :: Color t => Double -> t -> Vector -> LightSource t* - *makeDirectional i c d* tworzy kierunkowe źródło światła o intensywności i oraz kolorze c, świecące w kierunku wyznaczonym przez wektor d

- *makeSpherical :: Color t => Double -> t -> Vector -> LightSource t* - *makeSpherical i c x* tworzy kierunkowe źródło światła o intensywności i oraz kolorze c, znajdujące się w punkcie x

- *makeShadowRay :: Vector -> LightSource t -> Vector -> Ray* - *makeShadowRay n s x* tworzy promień pozwalający ustalić, czy punkt x o wektorze normalnym powierzchni n znajduje się w cieniu podczas rozważania źródła światła s

- *getLight :: Color t => LightSource t -> Vector -> Vector -> t* - *getLight s x n* zwraca ilość światła padającego na punkt x, o wektorze normalnym powierzchni n, ze źródła s

- *lIntersect :: LightSource t -> Double -> Vector -> Bool* - sprawdza, czy dany obiekt blokuje światło dla drugiego obiektu, podanego przez stałą t. Funkcja zakłada, że oba obiekty przecinają ten sam promień

4.3 Scene.hs

Plik zawierający funkcje przeprowadzające renderowanie sceny

- *render :: Color t => Scene t -> Image t* - główna renderująca scenę

- *Surface t* - typ danych reprezentujący powierzchnie, wykorzystujący reprezentację kolorów t.
  Konstruktory:

  - *Diffusive t* - powierzchnia rozpraszająca światło, o określonym kolorze

  - *Reflective* - powierzchnia odbijająca światło
  - *Luminous t* - powierzchnia świecąca własnym światłem, o określonym kolorze
  - *Mixed [(Double, Surface t)]* - powierzchnia powstała poprzez zmieszanie powyższych rodzajów powierzchni w określonych proporcjach

- *Object t* - typ danych reprezentujący obiekty sceny.
  Konstruktor:

  - *Object { geometry :: Geometry, surface :: Surface t }* - obiekt zdefiniowany przez jego kształt i rodzaj powierzchni

- *Scene t* - typ danych przechowujący informacje o scenie.
  Konstruktor:

  - *Scene { pxWidth :: Int, pxHeight :: Int, scrWidth :: Double, scrHeight :: Double, depth :: Double, bgColor :: t, rayDepth :: Int, lights :: [LightSource t], objects :: [Object t] }* - zbiór informacji opisujących scenę. Są to kolejno: wymiary gotowego obrazu, wymiary płótna na scenie, odległość ogniska od płótna, kolor tła, maksymalna głębokość rekursji dla promieni odbitych, lista źródeł światła oraz lista obiektów

- *Image t* - typ danych reprezentujący gotowy obraz.
  Konstruktor:

  - *Image { imWidth :: Int, imHeight :: Int, imPixels :: [t] }* - obraz opisany przez jego wymiary oraz zbiór pikseli w wybranej reprezentacji koloru

- *closestIntersect :: Ray -> [Object t] -> Maybe (Double, Object t)* - znajduje najmniejsze $t > 0$ takie, że $x + td$ jest punktem przecięcia danego promienia z pewnym obiektem z danej listy oraz obiekt, z którym przecięcie w tym punkcie następuje. Gdy takie t nie istnieje, zwracana jest wartość *Nothing*

- *traceRay :: Color t => Int -> t -> [LightSource t] -> [Object t] -> Ray -> t* - śledzi promień w celu ustalenia koloru badanego punktu obiektu. Jej argumenty to kolejno: maksymalna głębokość rekursji, kolor tła, lista źródeł światła, lista obiektów oraz badany promień

- *traceShadow :: Color t => LightSource t -> Vector -> Vector -> [Object t] -> Ray -> t* - funkcja śledząca promień w celu ustalenia, czy dany punkt znajduje się w cieniu. Jej argumenty to kolejno: badane źródło światła, rozważany punkt, wektor normalny prostopadły do powierzchni w tym punkcie, lista obiektów oraz badane źródło światła

- *makeRays :: Scene t -> [Ray]* - wyznacza listę promieni odpowiadających pikselom gotowego obrazu. Ich śledzenie pozwala utworzyć ten obraz

4.4 SceneParser.hs

Plik zawiera definicję prostego języka dla biblioteki Parsec, używanego do wczytywania plików

- *parseScene :: String -> String -> Either String (Scene RGB)* - w przypadku sukcesu zwraca scenę opisaną w parsowanym pliku, w przeciwnym wypadku zwraca informację o błędzie
- *pInt :: Parser Int* - parsuje liczbę całkowitą
- *pDouble :: Parser Double* - parsuje liczbę całkowitą lub zmiennoprzecinkową, w pierwszym przypadku otrzymana wartość konwertowana jest do typu *Double*
- *pMin :: (Num t, Ord t) => t -> Parser t -> Parser t - pMin m p* zwraca większą spośród wartości m oraz wartości będącej wynikiem działania parsera p
- *pPositive :: (Num t, Ord t) => Parser t -> Parser t* - funkcja równoważna wyrażeniu *pMin 0*
- *pVector :: Parser Vector* - parsuje wektor trójwymiarowy
- *pRGB :: Parser RGB* - parsuje kolor w postaci RGB
- *pLight :: Parser (LightSource RGB)* - parsuje pojedyncze źródło światła
- *pLights :: Parser [LightSource RGB]* - parsuje listę źródeł światła
- *pObject :: Parser (Object RGB)* - parsuje pojedynczy obiekt
- *pObjects :: Parser [Object RGB]* - parsuje listę obiektów
- *pGeometry :: Parser Geometry* - parsuje kształt obiektu
- *pSurface :: Parser (Surface RGB)* - parsuje opis powierzchni obiektu
- *pScene :: Parser (Scene RGB)* - parsuje scenę

4.5 Main.hs

Plik zawiera główne funkcje programu związane z operacjami wejścia/wyjścia.

- *imageToBmp :: Color t => Image t -> BMP* - zamienia obraz w wewnętrznym formacie programu na obraz w formacie używanym przez bibliotekę *Codec.BMP*
- *showImage :: Color t => String -> Image t -> IO ()* - wyświetla dany obraz oraz zapisuje go w formacie BMP
- *main :: IO ()* - funkcja główna programu