Hell Trace manual
By Przemysław Pastuszka

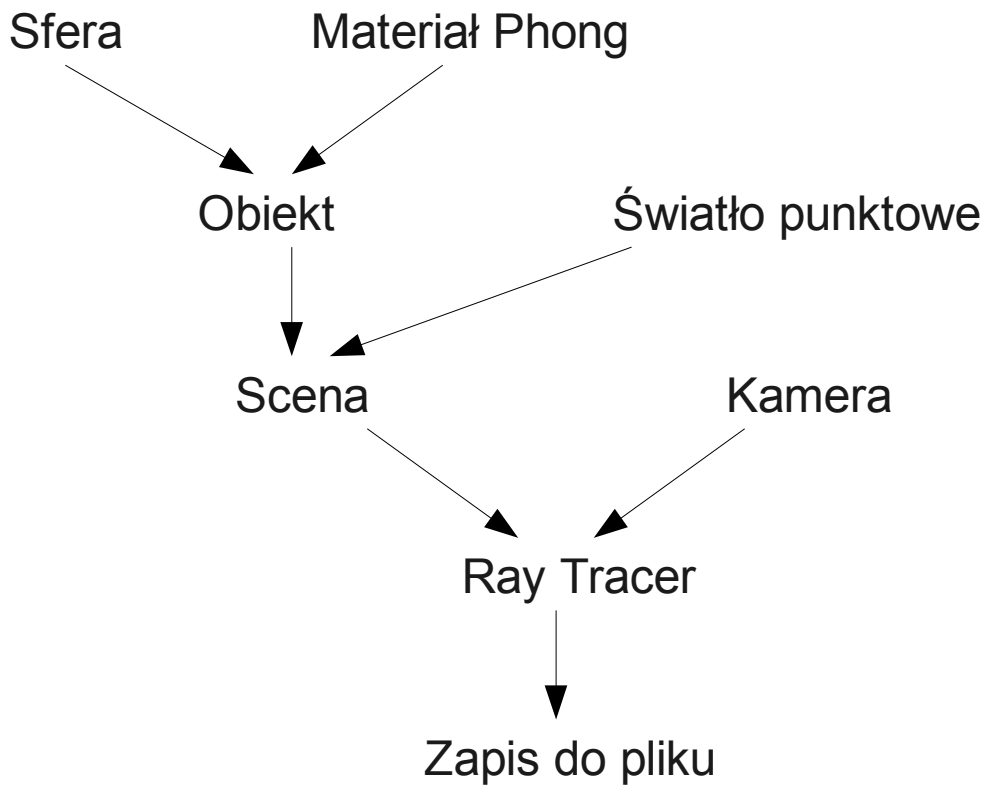
Poniższy manual został napisany z myślą o systemie Linux. Oczywiście, HellTrace działa również poprawnie na systemie Windows (na innych systemach nie testowałem), jednak może się zdarzyć, że pewne czynności opisane w tym manualu należy tam wykonać w zgoła inny sposób.

Aby rozpocząć korzystanie z HellTrace należy najpierw skompilować pliki źródłowe przy użyciu dołączonego pliku *makefile*, wywołując standardowe polecenie *make.* W ten sposób otrzymamy wykonywalny plik o nazwie *helltrace.* Wykonujemy go podając jako argument plik wejściowy z listą zadań, czyli: *./helltrace lista_zadan*

Skąd wziąć plik z listą zadań? Można pobrać przykładowy ze strony projektu, ale dużo lepiej i ciekawiej będzie stworzyć go samodzielnie. Czas więc na opis, jak to zrobić.

Najpierw muszę odpowiedzieć na pytanie: czym są wspomniane zadania? Otóż możemy na nie patrzeć jako na polecenia dla pewnych podprogramów programu HellTrace. Takie polecenie określa w jaki sposób podprogram ma działać (przekazuje atrybuty) oraz mówi, które z pozostałych zadań muszą zostać ukończone zanim rozpocznie się jego wykonanie. Gdy podprogram zakończy działanie, w środowisku powstaje nowy zasób (oprócz zadania - zapisz do pliku), np. nowy bufor pixeli lub nowe światło, który może zostać wykorzystany przez inne zadania.

Jak łatwo zauważyć, pomiędzy zadaniami tworzy się sieć powiązań. Przykładowo:



Jak łatwo zauważyć pierwszym zadaniem do wykonania będzie 'Sfera'. Uruchomi ono odpowiedni podprogram, który stworzy w środowisku sferę o zadanych atrybutach. Podobnie rzecz ma się z zadaniem 'Materiał Phong'. Ciekawsza rzeczy dzieją się w zadaniu 'Obiekt'. Odpowiedni podprogram tworzy obiekt, który jest złożeniem już wykonanej sfery i materiału. Dlatego zadanie 'Obiekt' musi zostać wykonane po zadaniach 'Sfera' i 'Materiał Phong'.

Nie ma znaczenia w jakiej kolejności w pliku pojawiają się zadania, gdyż menedżer zadań i tak zastosuje na nich sortowanie topologiczne. Ponadto wykonane zostaną tylko te zadania, które są wykorzystywane do wygenerowania pliku wyjściowego, tj. jeśli w grafie zależności nie istnieje ścieżka z danego zadania do zadania 'Zapis do pliku', to zadanie nie jest wykonywane.

W pliku wejściowym można tworzyć komentarze w stylu języka C++, tj.:
//zakomentowany tekst
oraz
/ zakomentowany*
tekst/*

Oto jak wygląda opis zadania:

```
TypZadania NazwaZadania
{
    [input] typAtrybutu1 nazwaAtrybutu1;
    ...
    [input] typAtrybutuN nazwaAtrybutuN;
}
```

TypZadania określa nazwę podprogramu do uruchomienia, zaś *NazwaZadania* to nazwa, jaką przyjmie zasób wygenerowany przez to zadanie. W nawiasach klamrowych znajduje się lista atrybutów z opcjonalnym słowem *input*. Jeśli *input* wystąpiło przed atrybutem *x* znaczy to, iż zasób *nazwaAtrybutuX* jest wymagany do wykonania zadania *NazwaZadania*. Niektóre z atrybutów w niektórych typach zadań są opcjonalne, tj. gdy zostaną pominięte, program użyje wartości domyślnej.

Oto lista możliwych typów zadań, łącznie z atrybutami i ich opisem.

-**Sphere** - tworzy prymityw, sferę. Uwaga: prymityw to jeszcze nie obiekt, który może zostać wykorzystany w scenie!
Atrybuty:
*origin - określa środek sfery. Przykładowe użycie: *origin 0,2,1;*
*radius - określa promień sfery
-**Plane** -tworzy prymityw, płaszczyznę
*normal - wektor normalnej płaszczyzny
*distance - odległość od punktu (0,0,0)
-**Phong** - tworzy materiał z wykorzystaniem cieniowania blinn-phonga
*color - kolor materiału (biel to 1,1,1)
*specColor - kolor odbicia światła (wartość domyślna: 1,1,1)
*diffuse - współczynnik rozproszenia światła

*reflectance - współczynnik odbicia
*specPower - współczynnik określający zachowanie się odbicia światła (domyślnie: 100)
-**PointLight** - światło punktowe
*color - wiadomo
*position - też wiadomo
*primitive - jeśli światło punktowe ma być widoczne na scenie jako obiekt, należy przekazać również prymityw (wykorzystujemy już stworzony zasób - proszę pamiętać o słowie input!). W przypadku pominięcia tego atrybutu, światło nie będzie widoczne jako obiekt (choć ciągle będzie mogło oświetlać scenę)
-**AreaLight** - światło powierzchniowe
*color - wiadomo
*primitive - wiadomo
*points - światło powierzchniowe jest przybliżane pewną ilością świateł punktowych. Im ich więcej, tym lepsza jakość końcowego oświetlenia, lecz również niższa wydajność raytracera. Przy pomocy tego atrybutu możesz zadać liczbę punktów, którymi światło powierzchniowe zostanie przybliżone
-**Simple** - prosty obiekt
*primitive - bryła
*material - materiał
-**Scene** - scena
*size - rozmiar sceny
*object1
...
*objectN - atrybuty od object1 do objectN określają kolejne obiekty, które wchodzą w skład sceny (w tym wypadku mogą być to światła lub obiekty typu Simple)
-**ConicCam** - kamera stożkowa
*width - szerokość w pikselach
*height - wysokość w pikselach
-**RayTracing -** wykonuje raytracing, tworząc bufor pikseli
*camera - wiadomo
*scene - wiadomo
*tracingDepth - jak długo (tj. ile odbić) może żyć promień (domyślnie: 5)
*width - szerokość wynikowego bufora
*height - wysokość wynikowego bufora
-**ImageLoader -** tworzy bufor pikseli na podstawie pliku
*fileName - wiadomo
*format - obsługiwane formaty to raw (wtedy należy podać również wysokość i szerokość wczytywanego obrazka), bmp oraz hlt (własny format pliku). Domyślnie: hlt
*width
*height - należy podać tylko w przypadku korzystania z odczytu pliku raw
-**ImageSaver -** zapisuje bufor pikseli do pliku
*buffer - bufor wejściowy
*fileName
*format - znów wiadomo (i domyślnie hlt)

-**SimpleToneMap** - tworzy nowy bufor pikseli, aplikując funkcję ekspozycji do wejściowego
*buffer - bufor wejściowy
*exposure - wartość ekspozycji (domyślnie 1)
-**AdvToneMap** - tworzy nowy bufor pikseli, aplikując operator Reinharda do wejściowego
*buffer - wiadomo
*middleGrey - punkt szarości (domyślnie: 0.6)
*whitePoint - punkt bieli (domyślnie: 16)
-**GaussianBlur** - nowy bufor, aplikuje filtr gaussa do starego
*buffer
*sigma - odchylenie standardowe
-**Lerp -** liniowa interpolacja, składa nowy bufor z dwóch wejściowych
*buffer1
*buffer2
*value - wartość z przedziału [0,1], stanowi o wagach buforów w czasie interpolacji
-**DepthAntialiasing -** antyaliasing na podstawie głębokości (tworzy nowy bufor)
*buffer
*treshold - jak stosunkowo duża musi być różnica głębokości, by zaaplikować antyaliasing (domyślnie: 1.2)
-**BrightPass -** kompresja ciemnych obszarów (tworzy nowy bufor)
*buffer
*middleGrey
*whitePoint - patrz AdvToneMap
*offset - domyślnie 1
*treshold - domyślnie 0.5