Grafika 3D - Etap 2

W tym etapie dostępna jest lista różnorodnych efektów, do zaimplementowania których nie są konieczne shadery (zależy od wybranej technologii). Należy wybrać dowolny zestaw efektów i zrealizować w scenie z pierwszego etapu tak, aby zdobyć odpowiednią ilość punktów. Maksymalnie można uzyskać w tym etapie 20 punktów, więc nie ma konieczności realizowania wszystkich zadań.

W przypadku przekroczenia 20 punktów, nadwyżka (**nie więcej niż 10** punktów) zostaje przeniesiona na konto etapu trzeciego.

Teksturowanie – wstęp

Teksturowanie pozwala na nałożenie pewnego wzoru na obiekt trójwymiarowy, dzięki czemu zyskuje on na szczegółowości.
Aby oteksturować model należy wyznaczyć tzw. współrzędne tekstury (proces ten nazywa się mapowaniem tekstur). Wierzchołek oprócz wielkości takich jak np. pozycja, wektor normalny czy styczny może posiadać jeden lub więcej zestawów współrzędnych tekstury. Najczęściej spotyka się tekstury dwuwymiarowe; rzadziej trójwymiarowe (mapy sześcienne) i jednowymiarowe.
Następnie dokonywane jest przekształcenie odwzorowujące piksele na ekranie w teksele z obszaru tekstury.

Stosując skrót myślowy, teksturowanie to przeprowadzenie procesu nałożenia obrazka na obiekt 3D. Współrzędne tekstury definiują punkty, które muszą się zgodzić w przestrzeni 2D tekstury oraz przestrzeni 3D obiektu.

Ponieważ nie zawsze pojedynczy piksel jednoznacznie odpowiada pojedynczemu tekselowi, wprowadzony został mechanizm filtrowania tekstur. Podstawowe filtrowanie realizowane jest automatycznie - zadaniem programisty jest jedynie włączenie i ustawienie wybranego typu filtrowania.

Trzy rodzaje filtrów:

- **Filtr powiększenia (MAGFILTER, MAGNIFICATION FILTER)**
  - Stosowany jest w przypadku, gdy obraz piksela jest mniejszy od rozmiaru teksela.
- **Filtr pomniejszenia (MINFILTER, MINIFICATION FILTER)**
  - Stosowany jeśli obraz piksela jest większy od rozmiaru teksela
- **Filtr mipmap (MIPFILTER, MIPMAP FILTER)**
  - Stosowany w przypadku używania mipmap. Technika wykorzystująca mipmapy używa kopii tej samej tekstury w różnych rozmiarach (poziomach szczegółowości). W zależności od odległości obiektu od kamery wybierany jest odpowiedni poziom mipmapy – najmniejszy zapewniający wystarczającą dokładność lub przeprowadzane jest filtrowanie sąsiednich poziomów.

Ustawienia filtrowania:

- **Wybranie najbliższego sąsiada (nearest-neighbour interpolation)**
  - To najprostsza i najszybsza metoda dająca najgorszy wynik wizualny. Wybierany jest najbliższy teksel lub najbliższa mipmapa bez żadnego mechanizmu mieszania kolorów. W przypadku używania najbliższego sąsiada może pojawić się duża liczba artefaktów i wyraźny aliasing. W różnych API nazywane często NONE, NEAR, NEAREST.

- **Filtrowanie biliniowe**
  - Często w API graficznych nazywane po prostu liniowe (w domyśle biliniowe).
    Jest to średnia ważona z czterech sąsiednich tekseli, pomiędzy którymi znajduje się dokładna pozycja przekształconego piksela. Innymi słowy jest to złożenie dwóch interpolacji liniowych – w kierunku U oraz w kierunku V.

- **Filtrowanie trójliniowe**
  - W przypadku używania mipmap jest to złożenie filtrowania biliniowego (w kierunkach U,V) z dodatkowym użyciem poziomów mipmap.
    W przypadku używania trójwymiarowych tekstur jest to złożenie filtrowania liniowego w trzech kierunkach U,V,W.
- **Filtrowanie anizotropowe**
  - Najwyższej jakości filtrowanie, bo dodatkowo opisane pewną funkcją zależną od kierunku względem obserwatora. Szczegóły wykraczają poza temat laboratorium.

Tekstury wykorzystywane są nie tylko do przechowywania koloru powierzchni – mogą np. przechowywać:
- wektory normalne (*normal map*),
- wartości współczynnika odbicia soczewkowego (*specular map*)
- wysokości punktów (*heightmap*)
- lokalne przesunięcia, wybrzuszenia punktów na płaskiej powierzchni (*displacement map*, *bumpmap*)
- informacje o cieniach (*shadowmap*)
- dowolne wartości ułożone w siatkę – np. współrzędne macierzy

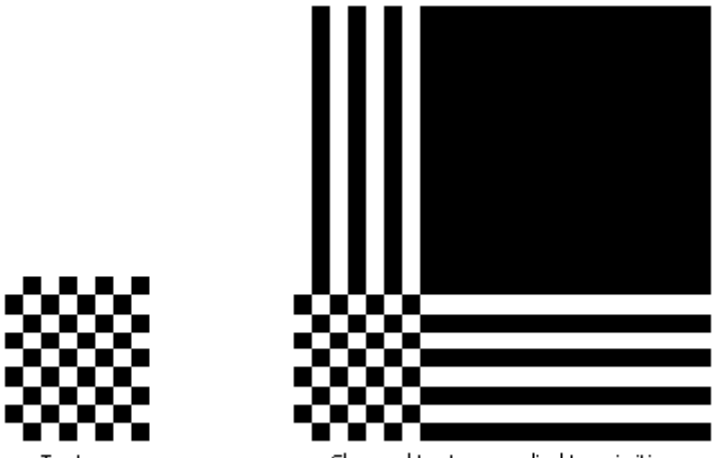
i wiele innych...

Dodatkowe uwagi:
- Zaleca się używanie tekstur, których wymiary są potęgami dwójki.
- Podanie współrzędnych tekstury z zakresu [0.0...1.0] powoduje rozciągnięcie tekstury na danym fragmencie siatki.
- Podanie współrzędnych z większego zakresu jest poprawne i powoduje zapętlenie kolejnych tekstur sąsiadująco. Efekt zależy od ustawionego trybu adresowania. Dostępne tryby adresowania to np.:

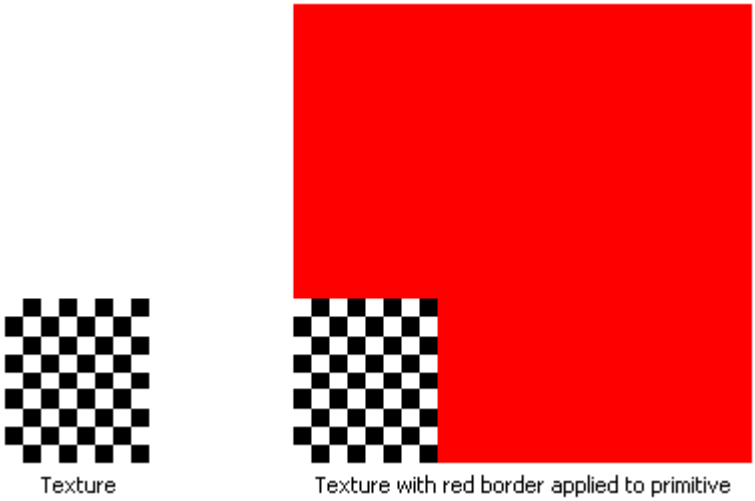
- Wrap (proste powtarzanie wzoru, przykład dla współrzędnych z zakresu [0.0...3.0])

- Mirror

- Clamp



- Border Color (wybranie dowolnego koloru)



Lista zadań

Uwaga: niektóre zadania opatrzone są dodatkową informacją:
[wymaga zadania nr N] - nie można zaliczyć zadania bez zrobienia również zadania nr N
[wyklucza zadanie nr N] – jeśli dwa zadania wzajemnie się wykluczają to można otrzymać punkty tylko za jedno z nich.

A. Podstawowe teksturowanie (max 6 punktów)

Zadanie 1. [**+4 punkty**]

Wybrać dowolne dwa obiekty na scenie i nałożyć na nie teksturę wczytaną z pliku obrazu w dowolnym formacie (np. *.PNG). Mogą to być nowo dodane obiekty.

Zadanie 2. [**+1 punkt**] [wymaga zadania nr 1.]

Dodać interfejs umożliwiający:

a) włączanie i wyłączanie liniowego filtru dla powiększenia
b) włączanie i wyłączanie filtrowania trójliniowego dla mipmap.
c) zmianę poziomu mipmap podczas teksturowania – spowoduje to rozmycie lub wyostrzenie obrazu na powierzchni obiektu. Słowa kluczowe: GL_TEXTURE_LOD_BIAS, MipMapLevelOfDetailsBias

Zadanie 3. [**+1 punkt**] [wymaga zadania nr 1.]

Należy umożliwić włączanie i wyłączanie w trakcie działania programu wygładzania krawędzi przy pomocy multisamplingu (MSAA – MultiSampleAntiAliasing).

B. Multiteksturowanie i mieszanie alfa (max 3 punkty)

Multiteksturowanie polega na wykorzystaniu więcej niż jednej tekstury do obliczenia koloru powierzchni w danym punkcie. Składanie tekstur wykonywane jest dynamicznie, co klatkę a nie w preprocessingu.
Mieszanie alfa polega na wykorzystaniu dodatkowej czwartej składowej koloru jako współczynnika w równaniu, które na podstawie koloru już zapisanego w buforze (destination color - *dst_color*, destination alpha - *dst_alpha*) oraz koloru właśnie rysowanego obiektu (source color - *src_color*, source alpha - *src_alpha*) obliczy nową wartość koloru w buforze.
W przypadku przezroczystości odpowiedni wzór to *src_alpha * src_color + (1-src_alpha) * dst_color*. Mieszanie addytywne z kolei to wzór postaci *src_alpha * src_color + dst_color*.
Trudna część realizacji przezroczystości związana jest z tym, że kiedy na ekranie nakłada się na siebie kilka warstw powierzchni przezroczystych nie można ich rysować w dowolnej kolejności. Aby otrzymać poprawny efekt trzeba je uporządkować według relacji zasłaniania.
W przypadku mieszania addytywnego kolejność rysowania obiektów jest dowolna.

Zadanie 4. [**+3 punkty**]

Należy oteksturować powierzchnię trawnika przy pomocy co najmniej dwóch tekstur. Zadanie może jednocześnie częściowo zaliczyć wykonanie zadania **nr 1** *(podłoga hali automatycznie spełnia warunek jednego z dwóch obiektów do oteksturowania)*.
Jedna tekstura to tekstura reprezentująca trawnik (np. trawa, ziemia itp.) – należy przygotować 2 wersje tej tekstury – np. różniące się nasyceniem kolorów.

Druga tekstura (lub kilka tekstur), która nałożona będzie na nawierzchnię, reprezentować powinna np. ścieżki, chodnik itp. W pozostałych miejscach niech posiada ustalony kolor lub kanał alfa ustawiony na 0. Nazwijmy tę teksturę **teksturą oznaczeń**.
Mieszanie tekstur polega na połączeniu tekstury materiału oraz tekstury oznaczeń.

W zadaniu ma istnieć możliwość łatwej podmiany tekstury nawierzchni w trakcie działania programu (wybrania jednej z dwóch dostępnych) – przy zachowaniu tekstury oznaczeń – dlatego łączenie musi odbywać się dynamicznie – w każdej klatce.

C. Generowanie i przekształcenia współrzędnych tekstury (max 6 punktów)

Zadanie 5. [**+2 punkty**]

Nałóż teksturę nawierzchni korzystając z jedynie współrzędnych XZ wierzchołków w układzie świata (przy założeniu, że Y to oś pionowa). Zamiast ręcznego podawania współrzędnych tekstury wykorzystaj macierz przekształcenia tekstury i dodaj interfejs umożliwiający skalowanie i przesuwanie nałożonej tekstury.
Nie ma konieczności, aby rozwiązanie działało jednocześnie z innymi efektami – można się przełączać lub przygotować dwie wersje skompilowanej wersji programu.

Zadanie 6. [**+4 punkty**]

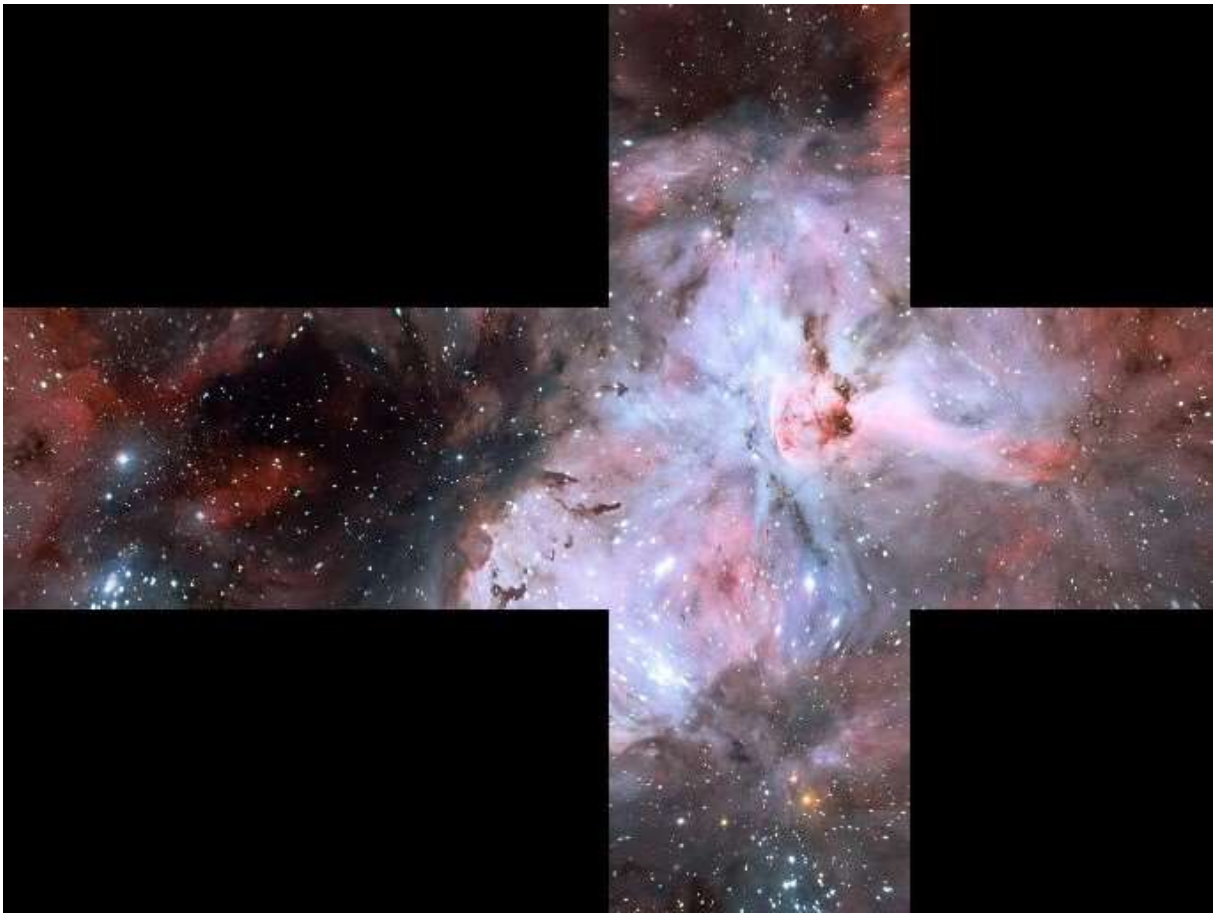
Jeden z reflektorów na scenie poza byciem źródłem światła niech posłuży za projektor. Zapewnij możliwość obracania go. Wykonaj rzutowanie perspektywiczne pewnej tekstury na pozostałe elementy sceny zgodnie z orientacją reflektora. Dla ułatwienia można wyłączyć wszystkie pozostałe tekstury na scenie.

W celu wyznaczenia współrzędnych tekstury dla obiektu, na który rzutowany jest obraz należy potraktować reflektor jako „kamerę”. Współrzędne punktu należy przekształcić podobnie jak przy renderowaniu, ale otrzymane współrzędne ekranowe „widziane” przez reflektor posłużą za współrzędne tekstury. Jeżeli podczas teksturowania obiektu wykorzystamy te właśnie współrzędne tekstury efektem będzie projekcja pewnego obrazu rzucanego przez reflektor na powierzchnię obiektu.

D. Tekstury sześcienne i proceduralne generowanie tekstur (max 10 punktów)

Zadanie 7. [**+2 punkty**] [wyklucza zadanie 8.]

Zadanie polega na dodaniu do sceny skyboxa. Jest to jedna z podstawowych technik wyświetlania nieba. Polega na utworzeniu sześcianu, który jest nieruchomy względem pozycji kamery (jeżeli kamera się przesuwa, to sześcian przesuwa się tak samo).
Idea jest taka, aby „zamknąć” obserwatora w pudełku, którego ścianki są bardzo daleko i przedstawiają niebo dookoła sceny. Teksturowanie skyboxa polega na dostarczeniu sześciu tekstur przedstawiających horyzont w sześciu kierunkach świata lub mapy sześciennej. Aby nie było widać złączenia krawędzi tekstury muszą idealnie pasować, ponadto na czas renderowania należy wyłączyć światło.



Zadanie 8. **[+3 punkty]** [wyklucza zadania 7.]

Zadanie jest identyczne z zadaniem nr 7. z tym, że należy użyć mapy sześciennej do renderowania skyboxa. Mapę sześcienną można utworzyć w kodzie z sześciu oddzielnych tekstur lub wczytać w specjalnym formacie do map sześciennych np. *.DDS (dla DirectX).

Zadanie 9. **[+4 punkty]**

Należy wygenerować dwuwymiarowy szum Perlina wewnątrz programu, a następnie na jego podstawie zmodyfikować siatkę trójkątów powierzchni trawnika (w razie potrzeby należy zagęścić siatkę powierzchni trawnika). Siatka powinna reprezentować trójwymiarowy wykres funkcji dwóch zmiennych, gdzie wartość funkcji odpowiada wartości szumu Perlina, nałożony na powierzchnię prostokąta, gdzie poszczególne wierzchołki przesuwane są w górę lub w dół w zależności od wartości funkcji.

E. Lustro oraz Mapowanie środowiskowe (max 12 punktów)

Lustro

Zrealizowanie poprawnego (nieprzybliżonego) efektu płaskiego lustra wymaga wykonania trzech przebiegów rysowania geometrii:
- najpierw rysowana jest oryginalna scena;
- następnie rysowana jest powierzchnia płaskiego lustra (test bufora głębokości musi być w tym przebiegu ustawiony na mniejsze lub równe, jeśli rysujemy drugi raz te same piksele; przed rysowaniem należy wyczyścić bufor szablonu i ustawić operację bufora szablonu tak, aby piksele lustra miały w nim ustawioną niezerową wartość; bufor szablonu – „stencil buffer” – to kolejny bufor obok bufora głębokości i buforów koloru bufor związany z kontekstem renderowania, więc tworząc kontekst renderowania trzeba upewnić się, że zostanie utworzony także bufor szablonu); - wreszcie rysowana jest cała scena jeszcze raz, ale odbita względem płaszczyzny lustra (przed rysowaniem trzeba wyczyścić bufor głębokości, oraz ustawić test bufora szablonu tak, aby rysowanie odbywało się tylko tam, gdzie widać piksele lustra, a więc gdzie w buforze szablonu jest niezerowa wartość; jeśli za lustrem znajdują się obiekty w oryginalnej scenie, to albo trzeba je zignorować w tym przebiegu, albo włączyć obcinanie rysowanej geometrii płaszczyzną lustra).
Można dodać ewentualnie czwarty przebieg i narysować półprzezroczystą powierzchnię lustra w określonym kolorze lub określoną teksturą (np. teksturą rys).

Przy rysowaniu odbitej sceny należy pamiętać o odbiciu również świateł. Odbicie geometrii (i świateł) zyskuje się poprzez ustawienie odpowiednio macierzy przekształcenia świata. Po wykonaniu tego dodatkowo zmieni się orientacja trójkątów, zatem trzeba zmodyfikować odpowiednio ustawienia „backface culling”. Szczególnej uwagi wymaga rysowanie billboardów odbitych w lustrze.

Zadanie 10. [**+4 punkty**]

Zrealizowanie podstawowego efektu lustra, zgodnie z powyższym opisem.

Zadanie 11. [**+1 punkt**] [wymaga zadania 10.]

Jeśli za lustrem znajduje się geometria i jest prawidłowo obcinana.

Zadanie 12. [**+1 punkt**] [wymaga zadania 10.]

Jeśli lustro jest półprzezroczyste i ma kształt elipsy.

Zadanie 13. [**+1 punkt**] [wymaga zadania 10. oraz 16.]

Jeśli w lustrze widać efekt cząsteczkowy z zadania nr 16.

Zadanie 14. [**+5 punktów**]

Umieść na scenie kulę i dokonaj mapowania środowiskowego (*Environmental Mapping*). Jest to efekt symulowania dynamicznych odbić zwierciadlanych środowiska na trójwymiarowym obiekcie. Szczegóły omawiane są na wykładzie a także dostępne w internecie.
Do wykonania tej techniki używana jest mapa sześcienna (*cubemap*, *cube texture*) zawierająca aktualny widok otoczenia we wszystkich kierunkach wokół obiektu.
Jeżeli rozwiązane zostało zadanie **nr 7 lub 8**, to wystarczy użyć mapy sześciennej, użytej do oteksturowani skyboxa.

http://en.wikipedia.org/wiki/Reflection_mapping
http://http.developer.nvidia.com/CgTutorial/cg_tutorial_chapter07.html

F. Billboardy / System cząsteczkowy (max 12 punktów)

Billboard to półprzezroczysta bitmapa nałożona na prostokąt, którego orientacja w przestrzeni dopasowuje się automatycznie do widoku kamery, podczas gdy położenie środka billboardu oraz jego rozmiary w układzie świata są stałe. W starych grach komputerowych wiele obiektów (które obecnie byłyby reprezentowane przez siatki trójkątów) było billboardami. Billboardy nadal są wykorzystywane do modelowania np. odległych obiektów (które nie muszą być dokładne) takich jak drzewa na horyzoncie, publiczności w grach sportowych, kęp trawy, chmur, słońca i w wielu innych sytuacjach.
W przypadku stosowania tej techniki należy pamiętać o poprawnym rozwiązaniu problemu przezroczystości, aby wyciąć tło z obrazu i wyświetlać jedynie obiekt. Istnieją dwa sposoby: test alfa (*alpha-test*) oraz mieszanie alfa (*alpha-blending*).
W przypadku testu alfa, w kanale alfa tekstury zapisana jest maska kształtu, który chcemy wyświetlić. Piksele, które nie przechodzą testu alfa nie są rysowane na ekranie ani uwzględniane w buforze głębokości. Brzeg kształtu uzyskany w ten sposób często jest poszarpany, ale nie ma konieczności wyświetlania obiektów w odpowiedniej kolejności. W przypadku alpha-blendingu konieczne jest sortowanie obiektów po współrzędnej głębokości w układzie kamery i rysowanie od najdalszych do najbliższych, ale efekt jest gładszy i bardziej atrakcyjny

Zadanie 15. [**+3 punktów**]

Umieszczenie na scenie publiczności w postaci kilkunastu lub kilkudziesięciu półprzezroczystych billboardów. Billboard reprezentuje np. źdźbła trawy lub liście drzew i jest zawsze zwrócony przodem do kamery.

Zadanie 16. [**+3 punkty**]

Utworzenie prostego efektu cząsteczkowego przy jednym z obiektów na scenie. Może on reprezentować ogień (np. wytwarzany przez ognisko), dym/parę wodna (wydostającą się na przykład z ust jednego z przechodniów) lub iskry (np. emitowane przez wadliwą latarnię). Efekt realizowany jest poprzez tzw. system cząsteczkowy.
Cząsteczki modelują mały, pojedynczy obiekt (np. płomienie, krople deszczu) lub grupę obiektów. Geometria cząsteczek to dwa trójkąty lub w niektórych API - sprzętowo wspierane PointSprites. Cząsteczki zachowują się jak półprzezroczyste billboardy – faktyczny kolor ma niezerową alfę, natomiast tło alfę zerową. Przykładowa tekstura cząsteczki wygląda tak:

Źródło (oraz przykład mieszania addytywnego i multiplikatywnego):
http://homepage.mac.com/nephilim/sw3ddev/additive_blending.html

Z cząsteczkami mogą być związane przeróżne właściwości fizyczne takie jak: kolor, rozbłysk, prędkość postępowa, prędkość kątowa (obracające się cząsteczki), przyspieszenie, masa, ciepło...
Zakładając, że współrzędna Y to kierunek wektora normalnego do powierzchni planetoidy w miejscu gdzie umieszczony jest emiter (umowny kierunek „do góry") w zadaniu wystarczy wprowadzić prostą animację cząsteczek polegającą na:

- tworzeniu cząsteczek w okolicy wybranego obiektu na scenie
- losowaniu wektora prędkości [X,Y,Z] bliskiego [0,1,0]. Cząsteczki wystrzeliwują w górę.
- po przebyciu ustalonej odległości cząsteczka ginie, a w jej miejsce tworzona jest nowa
- można ustalić limit cząsteczek (quota)
- ilość cząsteczek powinna być wartością łatwo sterowalną w kodzie

W ogólnym przypadku do kontrolowania cząsteczek służą dwa obiekty:

- ParticleEmitter – który stanowi źródło cząsteczek
- ParticleAttractor – obiekt, który wpływa na zachowanie cząsteczek

Systemy są szeroko stosowane w grafice 3D a także wizualizacjach fizycznych, automatów komórkowych itd.

http://en.wikipedia.org/wiki/Particle_system
http://www.youtube.com/watch?v=VWADjnwSQVA
http://www.youtube.com/watch?v=tuj4bGc5ZTk
http://www.youtube.com/watch?v=7wWsYTYGX78

Zadanie 17. [**+2 punktów**] [wymaga zadania 16.]

Zastosowanie **alpha-blending** i sortowanie cząsteczek względem kamery.

Zadanie 18. [**+1 punkt**] [wymaga zadania 16.]

Dopracowanie trajektorii cząsteczek:

- w początkowej fazie życia cząsteczki, dominują kierunki na boki (losowo promieniście)
- w przypadku ognia, cała objętość cząsteczek przypomina kształtem odwrócony tulipan
    - w środkowej fazie cząsteczka stopniowo zaczyna poruszać się do góry
    - w fazie końcowej zbiega ku osi środkowej
- w przypadku dymu cząsteczki stopniowo zaczynają poruszać się do góry, całość objętości cząsteczek przypomina paraboloidę.
- w przypadku iskier cząsteczki mają pewną masę i wpływa na nie grawitacja, więc każda z nich porusza się po krzywej parabolicznej zakrzywionej w kierunku podłogi.

Zadanie 19. [**+2 punkty**] [wymaga zadania 16.]

Tekstura cząsteczki jest animowana w czasie. Wystarczy przygotować jedną, większą teksturę zawierającą minimum 8 klatek animacji cząsteczki (sugerowane jest użycie przynajmniej 32 klatek)
W danym momencie wykorzystywany jest tylko fragment dużej tekstury zawierający bieżącą klatkę.

Zadanie 20. [**+1 punkt**] [wymaga zadania 19.]

W danym momencie **t** wynikowa tekstura cząsteczki jest interpolowana pomiędzy dwoma sąsiednimi klatkami względem czasu między wystąpieniem tych klatek.

G. Renderowanie do tekstury (max 12 punktów)

Do tej pory była mowa tylko o domyślnym procesie renderowania sceny do bufora koloru, który następnie przerzucany jest do bufora ramki i wyświetlany na ekranie.
Karty graficzne umożliwiają również ustawienie tekstury jako celu renderowania.
Technika często stosowana jest między innymi w tzw. renderowaniu odłożonym (*Deffered Shading*) oraz w połączeniu z prostokątem pełnoekranowym, na który nakładany jest obraz. Prostokąt jest niezależny od widoku i perspektywy i zawsze szczelnie wypełnia ekran. Dzięki wprowadzeniu tego dodatkowego etapu możliwe jest stosowanie przeróżnych filtrów i efektów graficznych w przestrzeni obrazu.
Stosowanie efektów po wyrenderowaniu sceny do tekstury i nałożeniu na ekran nazywane doczekało się nazwy „post-processing”.

Przydatne, przykładowe instrukcje:

C++ DirectX:

```
D3DXCreateTexture(pd3dDevice, width, height, 1, D3DUSAGE_RENDERTARGET,
       D3DFMT_A8R8G8B8, D3DPOOL_DEFAULT,&pOriginTexture);

SetRenderTarget(…)

RenderCopy(…)
```

Managed DirectX:

```
new Texture(device, width, height, 1, Usage.RenderTarget, Format.X8R8G8B8,
Pool.Default);
device.SetRenderTarget(…);

dodatkowo może się przydać obiekt klasy RenderToSurface
```

OpenGL:

W OpenGL warto skorzystać z instrukcji podobnych jak poniżej:

```
glGenFramebuffersEXT(1, &fbo);
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,
      GL_TEXTURE_2D, colorTex, 0);
glFramebufferTexture2DEXT(GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT,
      GL_TEXTURE_2D, depthTex, 0);
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT,...);
```

Na prawie wszystkich kartach graficznych można skorzystać z jednej prostej instrukcji:
`glCopyTexImage2D(...);`
W tym drugim przypadku odbywa się faktycznie kopiowanie z bufora koloru do tekstury, więc tekstura musi mieć identyczne rozmiary i format jak bufor koloru.
W OpenGL można również poradzić sobie bez renderowania do tekstury z pomocą bufora akumulacji, którego z kolei nie ma DirectX.

Zadanie 21. [**+3 punkty**]

Dodanie na scenie prostokąta symbolizującego ekran. Jedna jego strona teksturowana powinna być przy pomocy obrazu otrzymanego przez renderowanie do tekstury z pewnej konfiguracji kamery, dla której widać park.
Czyli w każdej klatce poza zwykłym obrazem, renderujemy także do tekstury (z ustalonego miejsca kamery) i wykorzystujemy wynik do teksturowania prostokąta.
Zadanie to można połączyć z zadaniem 5.

Zadanie 22. [**+4 punkty**]

Należy wyrenderować scenę do tekstury a następnie nałożyć na pełnoekranowy prostokąt. Naciśnięcie wybranego klawisza powoduje włączanie/wyłączanie efektu rozmycia gaussowskiego, którego stopień zależny od jasności piksela opisanej wzorem L = 0.3*R + 0.59*G+0.11*B. Rozmycie gaussowskie to filtr macierzowy – znany z przedmiotu Grafika 1. Im większa jasność – tym silniejsze rozmycie (większy promień uśredniania). Jest to uproszczona symulacja tzw. filtru Bloom.
Należy dodać wartość minimalną jasności (np. 120), którą można sterować (niekoniecznie musi być podczas działania programu – wystarczy w kodzie), poniżej której w ogóle nie następuje rozmycie.

Zadanie 23. [**+5 punkty**] [wymagane zadanie 14.]

Teksturę sześcienną użytą do mapowania środowiskowego lustrzanej kuli należy uzyskać poprzez renderowanie sceny do każdej z szczęściu tekstur mapy sześciennej. Poszczególne tekstury uzyskujemy ustawiając kamerę w środku kuli (skierowaną odpowiednio w górę/dół/lewo/prawo/przód/tył). Należy upewnić się, proporcje renderowanego obrazu (*aspect ratio*) wynoszą 1:1 (obraz jest kwadratowy) oraz, że kąt widzenia kamery w pionie i poziomie wynosi

90° (tak, aby na sześciu ścianach uzyskać pełny obraz sceny). Przy renderowaniu sceny do tych sześciu tekstur należy upewnić się, że mimo iż kamera znajduje się wewnątrz kuli, sama kula nie zasłania całej reszty sceny.

H. Mgła (max 3 punkty)

Zadanie 24. [**+3 punkty**]

API graficzne takie jak Direct3D, XNA oraz OpenGL umożliwiają włączenie liniowej lub ekspotencjalnej mgły.
Należy dodać dowolny rodzaj mgły do sceny wraz z możliwością włączenia/wyłączenia i zmiany gęstości/siły/odległości* z poziomu programu.

* - zależnie od tego, co dane API umożliwia

I. Płaszczyzny obcinania (max 4 punkty)

DirectX, XNA oraz OpenGL umożliwiają zdefiniowanie, zwykle do sześciu, płaszczyzn obcinania geometrii sceny („user clipping planes”). Tylko ta część geometrii sceny, która znajduje się po przedniej stronie aktywnych płaszczyzn obcinania jest rysowana. Do definiowania i włączania płaszczyzn obcinania w OpenGL służą polecenia: glEnable(GL_CLIP_PLANE0+i), glClipPlane(...). W DirectX Managed służy do tego struktura device.ClipPlanes.

Zadanie 25. [**+4 punkty**]

Płaszczyznę można zdefiniować jednoznacznie podając dowolny jej punkt i wektor prostopadły do niej (normalną). Niech w scenie zostanie wykorzystana płaszczyzna obcinania prostopadła do powierzchni boiska. Płaszczyznę można przesuwać w trakcie działania programu wzdłuż jednej z osi równoległych do boiska.
Część sceny znajdująca się po jednej stronie płaszczyzny obcinania powinna być rysowana normalnie, natomiast druga część jako wireframe (bez wypełniania trójkątów).
Jak łatwo się domyślić, zadanie wymaga dwóch przebiegów rysowania sceny ☺

J. GUI (max 6 punktów)

Zadanie 26. [**+6 punktów**] [wymaga zadania 2. oraz 3.]

Zadanie polega na stworzeniu GUI przy pomocy wyświetlania półprzezroczystych prostokątów, które nie są poddawane przekształceniom widoku i projekcji. Dodatkowo niech prostokąty mają zaokrąglone rogi – aby to zrealizować należy użyć kanału alfa (ponieważ tekstura musi być klasycznym prostokątem).
Pozycja i rozmiar prostokątów definiowane są we współrzędnych ekranu:
[0..1] x [0..1] lub [-1..1] x [-1..1] (zależnie od przyjętych ustaleń).

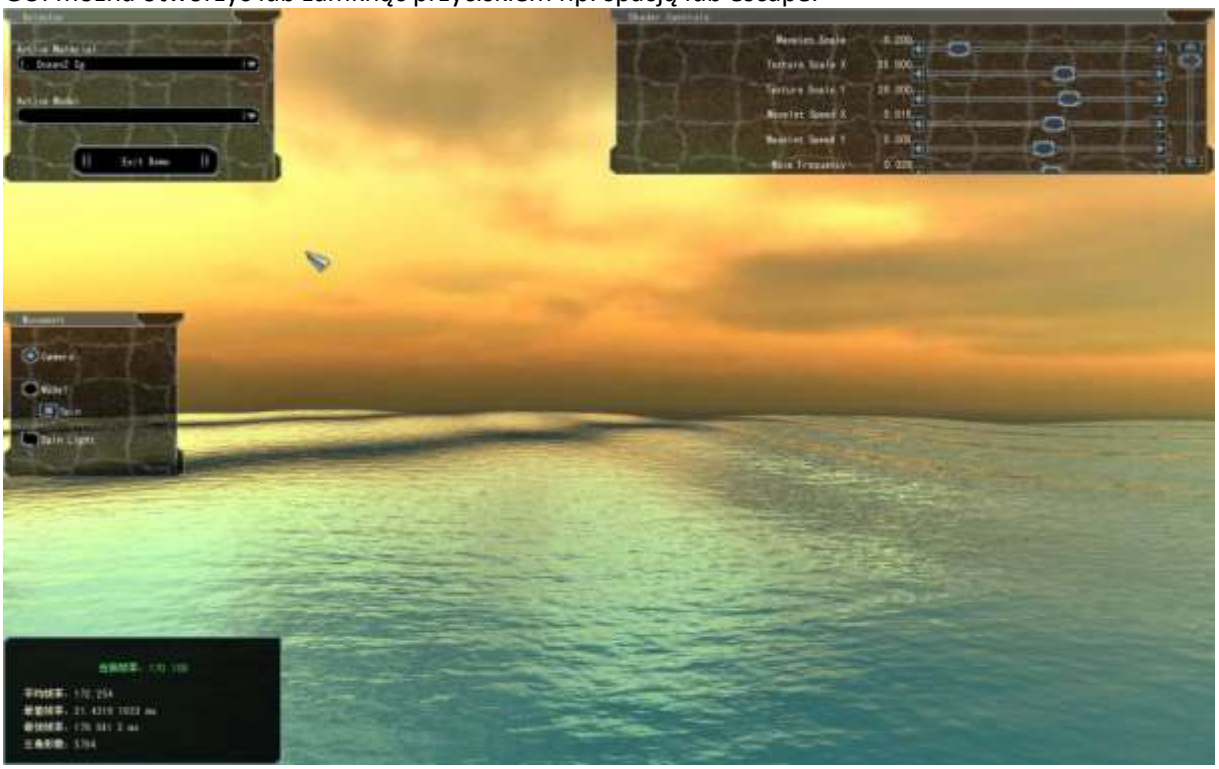
Gwoli ścisłości - przez rozmiar należy rozumieć prostokąt opisany na prostokącie z zaokrąglonymi rogami (czyli tak jak intuicja podpowiada).

Prostokąty zawsze zajmują tę samą część ekranu niezależnie od rozdzielczości i rozmiaru okna. Jeżeli przyjmiemy notację [0..1] x [0..1] oraz przykładowy prostokąt ma pozycję (0.2, 0.2) i wymiary (0.5, 0.5) to zawsze umieszczony jest w punkcie leżącym na 20% długości i szerokości a długość i szerokość ma równe 50% ekranu.
Należy umożliwić łatwą modyfikację rozmiarów prostokątów z poziomu kodu lub w pliku wejściowym. Warstwa graficzno-estetyczna nie jest oceniana i nie musi być dopracowana.

Minimalna funkcjonalność GUI to:

- interfejs dla zadań 2 i 3
- listBox z wyborem rozdzielczości ekranu (co najmniej 3 **różne** rozdzielczości do wyboru; mogą być z góry ustalone np. 1440x900, 1900x1080, 1900x1200.
  Po wybraniu danej rozdzielczości, zmiana jest aplikowana w programie.

GUI można otworzyć lub zamknąć przyciskiem np. spacją lub escape.



[Przykład – GUI w silniku OGRE]

Wszelkie pytania, wątpliwości, prośby o dokładniejsze wyjaśnienia konkretnych zadań proszę kierować na adres mailowy: P.Aszklar@mini.pw.edu.pl

Termin oddania projektu: 9. grudnia 2015r.