Ćwiczenie 6

OpenGL - teksturowanie powierzchni obiektów

1. Cel ćwiczenia

Celem ćwiczenia jest pokazanie podstawowych technik teksturowania powierzchni obiektów z wykorzystaniem mechanizmów biblioteki *OpenGL* z rozszerzeniem *GLUT*. Na przykładach zilustrowana będzie droga od przeczytania obrazu tekstury, do nałożenia jaj fragmentów na poszczególne fragmenty modelu obiektu trójwymiarowego. Pokazane zostaną przykłady teksturowania trójkąta, wielościanu i bardziej skomplikowanego modelu w postaci siatki trójkątów.

2. Teksturowanie powierzchni wieloboku

Teksturowanie polega na nanoszeniu na powierzchnie elementów modelu sceny obrazów zadanych w postaci map bitowych. Obrazy te zwykle umieszczone są w osobnych plikach. Ogólnie teksturowanie sprowadza się do trzech czynności. Po pierwsze odczytania z pliku obrazu tekstury i umieszczeniu odpowiednich danych w pamięci. Po drugie zdefiniowania tekstury, przez co należy rozumieć określenie sposobu interpretacji odczytanych z pliku danych i wreszcie po trzecie nałożenia tekstury na elementy modelu obiektu. Biblioteka OpenGL oferuje zestaw funkcji realizujących dwa ostatnie kroki procedury, czyli definiowanie i nakładanie tekstur. Funkcji realizujących odczytywanie obrazów z plików graficznych i umieszczanie ich w pamięci w bibliotece OpenGL nie ma.

Na wstępie pokazany będzie prosty przykład ilustrujący zagadnienie teksturowania powierzchni trójkąta. W ćwiczeniu 3 napisany został program wyświetlający obraz modelu obracającego się jajka (można także wykorzystać napisany w ćwiczeniu 4 program pozwalający obserwować jajko z punktu na powierzchni sfery). Dla ilustracji problemów jakie występują w procesie teksturowania w programie tym należy dokonać kilku zmian:

- W miejsce modelu jajka należy umieścić trójkąt.
- Wykorzystując wiadomości z ćwiczenia 5 oświetlić trójkąt. W tym celu należy zdefiniować własności materiału z jakiego wykonany jest trójkąt i wprowadzić na scenę punktowe źródło światła. Najlepiej aby trójkąt i światło miały kolor biały. Otrzymany rezultat powinien wyglądać mniej więcej tak jak to pokazano na rys. 1.

Rys. 1. Biały trójkąt oświetlony białym światłem

- Program uzupełnić dodając poniżej zapisany kod funkcji służącej do odczytywania obrazu tekstury z pliku graficznego w formacie TGA (targa). Jeśli dysponuje się innymi funkcjami, pozwalającymi na odczytanie i zapisanie w pamięci danych o obrazach w innych formatach można je wykorzystać.

```
/*************************************************************************************/
 // Funkcja wczytuje dane obrazu zapisanego w formacie TGA w pliku o nazwie
 // FileName, alokuje pamięć i zwraca wskaźnik (pBits) do bufora w którym
 // umieszczone są dane.
 // Ponadto udostępnia szerokość (ImWidth), wysokość (ImHeight) obrazu
 // tekstury oraz dane opisujące format obrazu według specyfikacji OpenGL
 // (ImComponents) i (ImFormat).
 // Jest to bardzo uproszczona wersja funkcji wczytującej dane z pliku TGA.
 // Działa tylko dla obrazów wykorzystujących 8, 24, or 32 bitowy kolor.
 // Nie obsługuje plików w formacie TGA kodowanych z kompresją RLE.
/*************************************************************************************/
```

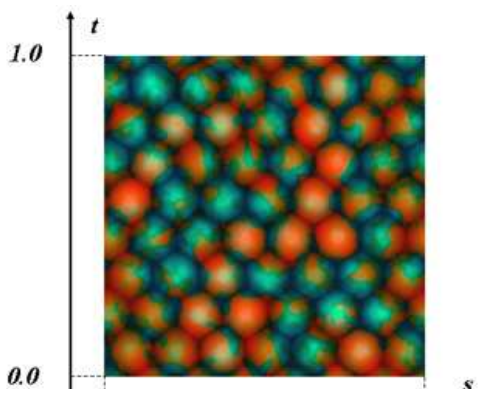
```
GLbyte *LoadTGAImage(const char *FileName, GLint *ImWidth, GLint *ImHeight, GLint *ImComponents, GLenum
*ImFormat)
{
/*****************************************************************************/
// Struktura dla nagłówka pliku  TGA

    #pragma pack(1)
    typedef struct
    {
        GLbyte    idlength;
        GLbyte    colormaptype;
        GLbyte    datatypecode;
        unsigned short    colormapstart;
        unsigned short    colormaplength;
        unsigned char     colormapdepth;
        unsigned short    x_orgin;
        unsigned short    y_orgin;
        unsigned short    width;
        unsigned short    height;
        GLbyte    bitsperpixel;
        GLbyte    descriptor;
    }TGAHEADER;
    #pragma pack(8)

    FILE *pFile;
    TGAHEADER tgaHeader;
    unsigned long lImageSize;
    short sDepth;
    GLbyte    *pbitsperpixel = NULL;

/*****************************************************************************/
// Wartości domyślne zwracane w przypadku błędu

    *ImWidth = 0;
    *ImHeight = 0;
    *ImFormat = GL_BGR_EXT;
    *ImComponents = GL_RGB8;

    pFile = fopen(FileName, "rb");
    if(pFile == NULL)
      return NULL;
/*****************************************************************************/
// Przeczytanie nagłówka pliku

    fread(&tgaHeader, sizeof(TGAHEADER), 1, pFile);

/*****************************************************************************/
// Odczytanie szerokości, wysokości i głębi obrazu

    *ImWidth = tgaHeader.width;
    *ImHeight = tgaHeader.height;
    sDepth = tgaHeader.bitsperpixel / 8;

/*****************************************************************************/
// Sprawdzenie, czy głębia spełnia założone warunki (8, 24, lub 32 bity)

    if(tgaHeader.bitsperpixel != 8 && tgaHeader.bitsperpixel != 24 && tgaHeader.bitsperpixel != 32)
        return NULL;

/*****************************************************************************/
// Obliczenie rozmiaru bufora w pamięci

    lImageSize = tgaHeader.width * tgaHeader.height * sDepth;

/*****************************************************************************/
// Alokacja pamięci dla danych obrazu

     pbitsperpixel = (GLbyte*)malloc(lImageSize * sizeof(GLbyte));

      if(pbitsperpixel == NULL)
        return NULL;

    if(fread(pbitsperpixel, lImageSize, 1, pFile) != 1)
        {
        free(pbitsperpixel);
        return NULL;
        }

/*****************************************************************************/
// Ustawienie formatu OpenGL

    switch(sDepth)
        {
        case 3:
            *ImFormat = GL_BGR_EXT;
            *ImComponents = GL_RGB8;
            break;
        case 4:
            *ImFormat = GL_BGRA_EXT;
            *ImComponents = GL_RGBA8;
            break;
        case 1:
            *ImFormat = GL_LUMINANCE;
            *ImComponents = GL_LUMINANCE8;
            break;
        };

    fclose(pFile);

    return pbitsperpixel;
}
/*****************************************************************************/
```

- W funkcji `MyInit()` dopisać kilka linii kodu zawierającego podstawowe funkcje definiujące właściwości procesu teksturowania, istotne dla mechanizmu renderującego OpenGL . Najważniejsze z tych funkcji to: `glTexImage2D()` służąca do definiowania tekstury dwuwymiarowej, `glTexEnvi()` określająca tak zwany tryb teksturowania i `glTexParameteri()` opisująca sposób nakładania tekstury na powierzchnie modeli obiektów. Rola tych funkcji i znaczenie ich argumentów zostaną obszerniej opisane w dalszej części instrukcji.

```
/*************************************************************************************/
// Funkcja ustalająca stan renderowania
/*************************************************************************************/

void MyInit(void)
{
/*************************************************************************************/
// Zmienne dla obrazu tekstury

 GLbyte *pBytes;
 GLint ImWidth, ImHeight, ImComponents;
 GLenum ImFormat;

          // ..................................
          //       Pozostała część funkcji MyInit()
          // ..................................

/*************************************************************************************/
// Teksturowanie będzie prowadzone tyko po jednej stronie ściany

 glEnable(GL_CULL_FACE);

/*************************************************************************************/
//  Przeczytanie obrazu tekstury z pliku o nazwie tekstura.tga

 pBytes = LoadTGAImage("tekstura.tga", &ImWidth, &ImHeight, &ImComponents, &ImFormat);

 /*************************************************************************************/
// Zdefiniowanie tekstury 2-D

 glTexImage2D(GL_TEXTURE_2D, 0, ImComponents, ImWidth, ImHeight, 0, ImFormat, GL_UNSIGNED_BYTE, pBytes);

/*************************************************************************************/
// Zwolnienie pamięci

 free(pBytes);

/*************************************************************************************/
// Włączenie mechanizmu teksturowania

 glEnable(GL_TEXTURE_2D);

/*************************************************************************************/
// Ustalenie trybu teksturowania

 glTexEnvi(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);

/*************************************************************************************/
// Określenie sposobu nakładania tekstur

 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
 glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

          // ..................................
          //       Pozostała część funkcji MyInit()
          // ..................................

}
/*************************************************************************************/
```

- Udostępnić obraz tekstury zapisany w pliku graficznym w formacie TGA. Plik **tekstury.zip** zawiera kilkanaście przykładów tekstur o rozmiarach 256 na 256 pikseli i 24 bitowej informacji o kolorze piksela. Do dalszych eksperymentów należy wybrać jedną z proponowanych tekstur lub wykorzystać własny wzorzec zapisany w takim samym formacie.

- Na zakończenie w funkcji definiującej obiekt (trójkąt) należy wprowadzić współrzędne wzorca tekstury definiujące fragment obrazu, jaki ma być naniesiony na powierzchnię trójkąta. Przykładowy obraz tekstury i związny z nim układ współrzędnych pokazane zostały na rysunku 2.

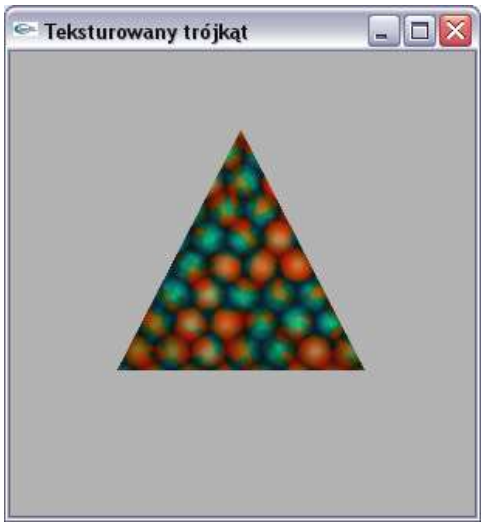


Rys. 2. Wzorzec w układzie współrzędnych tekstury

Kod pozwalający na przyporządkowanie fragmentów przestrzeni tekstury elementom modelu jest bardzo prosty. Funkcja **glTexCoord2f()** poprzez swoje argumenty określa, które wierzchołki trójkąta naniesionego na wzorzec tekstury odpowiadają , którym wierzchołkom teksturowaneg trójkąta.

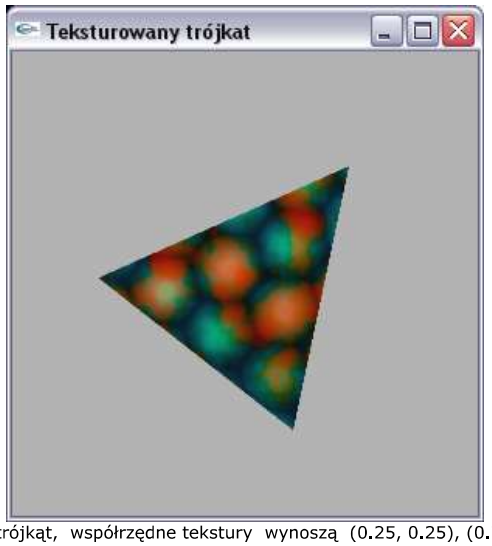
```
glBegin(GL_TRIANGLES);

        glNormal3f(...);
        glTexCoord2f(0.0f, 0.0f);
        glVertex3f(...);

        glNormal3f(...);
        glTexCoord2f(1.0f, 0.0f);
        glVertex3f(...);

        glNormal3f(...);
        glTexCoord2f(0.5f, 1.0f);
        glVertex3f(...);

glEnd();
```

Ostateczny efekt naniesienia fragmentu obrazu z rysunku 2 na powierzchnię trójkąta z rysunku 1 jest następujący:



Rys. 3. Teksturowany trójkąt, współrzędne tekstury wynoszą (0.0, 0.0), (1.0, 0.0), (0.5, 1.0)

Jeśli zmienić współrzędne wzorca tekstury na przykład na (0.25, 0.25), (0.75, 0.25), (0.5, 0.75) obraz umieszczony na powierzchni trójkąta będzie oczywiście inny. Przypadek taki został zilustrowany na rysunku 4.



Rys. 4. Teksturowany trójkąt, współrzędne tekstury wynoszą (0.25, 0.25), (0.75, 0.25), (0.5, 0.75)

Należy zwrócić jeszcze uwagę na bardzo istotny element procesu teksturowania. Można zaobserwować jak działa, występująca w funkcji `MyInit()` funkcja `glEnable(GL_CULL_FACE)`. Gdy tej funkcji nie ma teksturowanie prowadzone jest dla obu stron wieloboku, gdy funkcja jest teksturowana jest jedna strona. Która strona wieloboku jest teksturowana zależy od kolejności podania wierzchołków w jego opisie. Zastosowanie funkcji `glEnable(GL_CULL_FACE)` pozwala na uniknięcie prowadzenia obliczeń tekstury w przypadku niewidocznych wnętrz obiektów.

3. Definiowanie tekstury i sterowanie procesem teksturowania

W poprzednim punkcie wspomniano, że za przebieg porcesu teksturowania odpowiedzialne są głównie trzy funkcje umieszczone w funkcji `MyInit()`. Są to funkcje: `glTexImage2D()`, `glTexEnvi()` i `glTexParameteri()`. Omówiona zostanie teraz kolejno ich rola.

- Funkcja `glTexImage2D()` definiuje teksturę dwuwymiarową i określona jest jako:

```
void glTexImage2D(GLenum target, GLint level, GLint components, GLsizei width, GLsizei height, GLint
                  border, GLenum format, GLenum type, const GLvoid *pixels)
```

Argumenty mają następujące znaczenie:

Argument	Opis
target	Rodzaj definiowanej tekstury, zawsze powinno być **GL_TEXTURE_2D**
level	Poziom szczegółowości. Jest ustawiany na 0 gdy ***mipmapy*** nie są stosowane, gdy ***mipmapy*** są używane ***level*** jest liczbą określająca poziom redukcji obrazu tekstury.
components	Ilość używanych składowych koloru, liczba od 1 do 4.
width	Szerokość obrazu tekstury, zawsze jest potegą liczby 2, może być powiększona o szerokość tak zwanej ramki tekstury
heigth	Wysokość obrazu tekstury, zawsze jest potęgą liczby 2, może być powiększona o szerokość ramki tekstury.
border	Szerokość ramki tekstury, może wynosić 0, 1 lub 2
format	Format danych obrazu tekstury, określa co opisują dane, czy są indeksami kolorów czy ich składowymi (np. R, G, B) i w jakiej są podane kolejności.
type	Typ danych dla punktów obrazu tekstury, określa jak kodowane są dane, istnieje możliwość używania różnych formatów liczb, od 8 bitowych liczb stałoprzecinkowych, do 32 bitowych liczb zmiennoprzecinkowych.
piksels	Tablica o rozmiarze ***width*** x ***height*** x ***components***, w ktorej znajdują się dane z obrazem tekstury.

Szczegółowe informacje o możliwych ustawieniach poszczególnych argumentów (dotyczy to w szcególności argumentów `format` i `type`) można znaleźć w dokumentacji OpenGL.

- Funkcja `glTexEnvi()` służy do ustalenia tak zwanego trybu teksturowania, czyli sposobu łączenia koloru piksela obrazu tekstury z kolorem piksela ekranu. Prototyp funkcji ma postać:

```
void glTexEnvi( GLenum target, GLenum pname, GLint param )
```

Argument `target` należy ustawić na `GL_TEXTURE_ENV`, a `pname` na `GL_TEXTURE_ENV_MODE`. Sposób łaczenia pikseli tekstury i ekranu określa trzeci z argumentów `param`, który może przyjmować cztery wartości:

`GL_MODULATE` - kolor piksela ekranu jest mnożony przez kolor piksela tekstury, następuje w ten sposób mieszanie barw tekstury i tego co jest teksturowane,

`GL_DECAL` - piksele tekstury zastępują piksele na ekranie,

`GL_BLEND` - kolor piksela ekranu jest mnożony przez kolor piksela tekstury i łączony ze stałym kolorem,

`GL_REPLACE` - działa podobnie jak `GL_DECAL` różnica występuje wtedy, gdy wprowadzona zostaje przezroczystość.

- Funkcja `glTexParameteri()` pozwala między innymi na określenie algortmów dodawania nowych piksel w przypadku, gdy w obrazie tekstury jest za mało punktów aby wypełnić obraz teksturowanego elementu oraz usuwania pikseli, gdy w obrazie tekstury jest ich za dużo. Umożliwia także określenie sposobu postępowania w przypadku, gdy zadane w programie współrzędne tekstury wykraczają poza zakres [0, 1]. Definicja funkcji wygląda następująco:

```
void glTexParameteri(GLenum target, GLenum pname, GLint param)
```

Argument `target` w przypadku tekstur dwuwymiarowych (tylko takich dotyczy niniejsze ćwiczenie) należy ustawić na `GL_TEXTURE_2D`. Pozostałe dwa argumenty `pname` i `param` dotyczą szczegółow działania algorytmów teksturowania. Zestawienie wartości tych parametrów zawiera poniższa tabela.

pname	***param***
GL_TEXTURE_MIN_FILTER	Opisuje w jaki sposób następuje pominiejszanie tekstury (usuwanie pikseli) **GL_NEAREST** - usuwa się najbliższy (w sensie metryki Manhattan) punkt sąsiedni

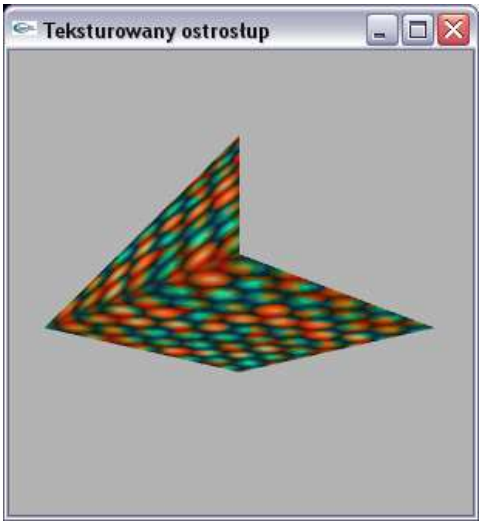
	GL_LINEAR - filtracja liniowa z odpowiednimi wagami

pname	*param*
GL_TEXTURE_MAG_FILTER	Opisuje w jaki sposób następuje powiększanie tekstury (dodawanie pikseli) **GL_NEAREST** - dodaje się najbliższy (w sensie metryki Manhattan) punkt sąsiedni **GL_LINEAR** - interpolacja liniowa wykorzystująca wartości sąsiednich pikseli
GL_TEXTURE_WRAP_S	Opisuje sposób traktowania współrzędnej ***s*** tekstury, gdy wykracza ona poza zakres [0, 1] **GL_CLAMP** - poza zakresem stoswany jest kolor ramki tekstury lub stały kolor **GL_REPEAT** - tekstura jest powtarzana na całej powierzchni wielokąta
GL_TEXTURE_WRAP_T	Opisuje sposób traktowania współrzędnej ***t*** tekstury, gdy wykracza ona poza zakres [0, 1] **GL_CLAMP** - poza zakresem stosowany jest kolor ramki tekstury lub stały kolor **GL_REPEAT** - tekstura jest powtarzana na całej powierzchni wielokąta

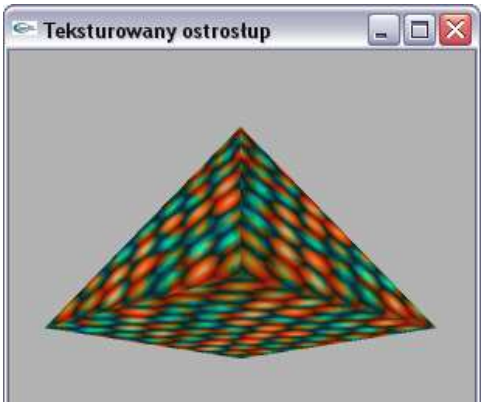
Szczegółowe informacje o funkcjonowaniu algorytmów wybieranych przy pomocy argumentów `pname` i `param` można zanleźć w dokumentacji *OpenGL*.

4. Teksturowanie wielościanu

Następne zadanie polega na napisaniu programu rysującego obraz piramidy, czyli ostrosłupa o podstawie kwadratu. Poszczególne ściany ostrosłupa maja być teksturowane tylko z jednej, widocznej strony. Jako szkielet należy wykorzystać napisany ćwiczeniu 3 program wyświetlający obraz modelu obracającego się jajka, lub wykonany w ćwiczeniu 4 program pozwalający obserwować jajko z punktu na powierzchni sfery. Przy pisaniu programu należy kontrolować prawidłowość rysowania kolejnych ścian aż do zamknięcia ostrosłupa. Aby można było sprawdzić prawidłowość rysowania obrazów ścian bryły, należy wprowadzić sterowanie widocznością poszczególnych ścian przy pomocy klawiszy, na przykład od 1 do 5. Obraz dwóch ścian piramidy pokazany został na rysunku 5 a cały wielościan na rysunku 6.



Rys. 5. Dwie ściany teksturowanego ostrosłupa

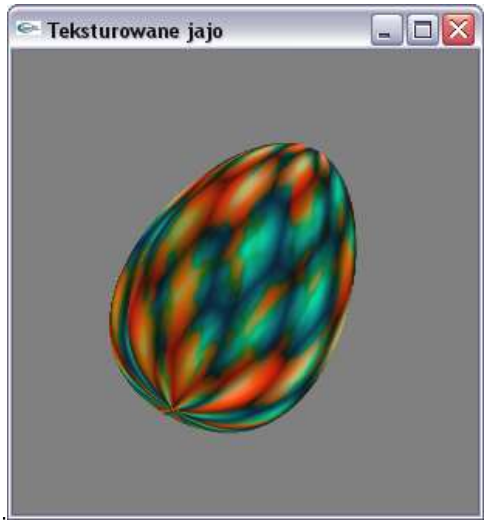


Rys. 6. Widok kompletnego ostrosłupa

5. Teksturowanie wielościanu, który powstał z powierzchni opisanej parametrycznie

W poprzednich ćwiczeniach rysowano obrazy powierzchni jajka. W ćwiczeniu 2 powierzchnia została zadana przy pomocy równań parametrycznych i zamieniona na model szkieletowy zbudowany z trójkątów. Algorytm polegał na równomiernym podziale dwuwymiarowej dziedziny parametrycznej na małe trójkąty i wyliczeniu przy pomocy zadanych równań współrzędnych wierzchołków trójkątów w przestrzeni 3-D. Taki sam sposób można wykorzystać przy teksturowaniu powierzchni opisanej parametrycznie. Wystarczy bowiem obraz tekstury pokryć taką samą siatką trójkątów jaką naniesiono na dziedzinę parametryczną i odpowiednio ustawić współrzędne punktów w przestrzeni tekstury przy pomocy funkcji `glTexCoord2f()`. Oczywiście równomierny podział przestrzeni tekstury spowoduje określone deformacje obrazu nałożonego na powierzchnię. Zadanie tak naprawdę polega bowiem na "naciągnięciu" kwadratowego obrazu na powierzchnię jajka.

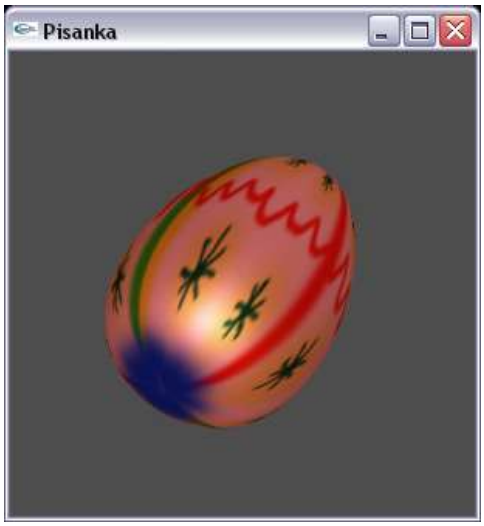
Znów jako szkielet programu można wykorzystać program z ćwiczenia 3 wyświetlający obraz modelu obracającego się jajka, albo napisany w ćwiczeniu 4 program, który pozwala obserwować jajko z dowolnego punktu na powierzchni sfery. Przed teksturowaniem najlepiej kolory jajka i światła ustawić na biały. Po nałożeniu tekstury przy pomocy wcześniej opisanej metody jajko powinno wyglądać tak jak pokazano to na rysunku 7.



Rys. 7. Jajko z nałożoną teksturą przy podziale obrazu tekstury takim samym jak podział dziedziny parametrycznej

Można przeprowadzić eksperyment polegajacy na zbadaniu jak dokładność podziału, czyli liczba trójkątów z których zbudowany jest model wpływa na wygląd obrazu teksturowanego obiektu. Interesujące jest także sprawdzić, jaki efekt uzyska się przez nakładanie tekstur o większej rozdzielczości np. 512 x 512 czy 1024 x1024 punktów. Do wykonania tego eksperymentu można wykorzystać tekstury: **t_256.zip**, **t_512.zip** i **t_1024.zip**.

Stwierdzono już wcześniej, że nałożenie tekstury propnowaną metodą prowadzi do deformacji jej obrazu. Widać więc, że ustalenie algorytmu podziału obrazu tekstury na fragmenty odwzorowywane następnie na na poszczególne elementy modelu nie jest zadaniem banalnym. Na pewno lepsze efekty można by uzyskać stosując podział obrazu tekstury na trójkąty o różnej wielkości. Ponadto nawet przy tak prostym algorytmie jaki został użyty w przypadku pokazanym na rysunku 7 wcale nie jest jasne, które fragmenty obrazu tekstury odwzorowywane są, na które fragment powierzchni jajka. Aby zbadać ten problem proponuje sie spróbować wykonać teksturowanie powierzchni jajka "malując" obraz w przestrzeni tekstury i sprawdzając jakie daje to efekty na obrazie teksturowanego jajka. Celem jaki należy osiągnąć może być stworzenie tekstury, która po nałożeniu na powierzchnię jajka da efekt "pisanki" na przykład podobnej do takiej jak na rysunku 8. Do eksperymentu można wykorzystac każdy program graficzny pozwalający na zapisanie efektu "malowania" w formacie TGA n.p. program GIMP.



Rys. 8. "Pisanka"

Ćwiczenie opracował: Jacek Jarnicki, korekta Marek Woda (2017-12-13)

Back