Efekt lustra 3D w OpenGL z wykorzystaniem bufora szablonowego (stencil buffer)

Autor: Radosław Płoszajczak

Spis treści

I. Wstęp

Ten tutorial opisuje jak wykonać lustro przy użyciu OpenGL z realistycznym odbiciem wybranych obiektów sceny. W przykładzie zostanie wykorzystany bufor szablonowy (stencil buffer) oraz płaszczyzna obcinająca (clip plane). Dla lepszego efektu obiekt odbijający scenę nie będzie idealnym lustrem lecz refleks zostanie zrealizowany na szybie pomalowanej w kolorową szachownicę.

II. Metoda rysująca przezroczystą szybę

Pierwszym etapem będzie utworzenie nowej metody w klasie `TForm1` rysującej kwadratową szybę, w której będą się odbijać obiekty sceny. Rysowanie realizuje poniższa funkcja w pliku *Unit1.cpp*:

```
void __fastcall TForm1::RysujSzybe()
{
      const float przezroczystosc = 0.3f;
      const int il_rzedow_kolumn = 8;
      const GLfloat a = 0.5f;

      glPushMatrix();
      glTranslatef(-a*il_rzedow_kolumn/2.0f, 0.0f, -a*il_rzedow_kolumn/2.0f);

      glNormal3f(0.0f, 0.0f, 1.0f);
      for(int i = 0; i < il_rzedow_kolumn; i++)
            for(int j = 0; j < il_rzedow_kolumn; j++)
            {
                  glBegin(GL_QUADS);
                  if((i + j) % 2)
                        glColor4f(1.0f, 0.0f, 0.0f, przezroczystosc); //czerwony
                  else
                        glColor4f(1.0f, 1.0f, 0.0f, przezroczystosc); //zolty

                  glVertex3f(i * a, 0.0f, (j + 1) * a);
                  glVertex3f((i + 1) * a, 0.0f, (j + 1) * a);
                  glVertex3f((i + 1) * a, 0.0f, j * a);
                  glVertex3f(i * a, 0.0f, j * a);
                  glEnd();
            }
      glPopMatrix();
}
```

Na początku zdefiniowane są stałe określające przeźroczystość szyby (wartość koloru alfa), ilość kolumn i rzędów szachownicy oraz wysokość i szerokość każdego pola `a`. Aby wyśrodkować rysowaną powierzchnię aktualna macierz model – widok jest ładowana na stos, a następnie układ współrzędnych przesuwany jest o połowę szerokości i wysokości całej szyby w kierunku ujemnych wartości osi X i Z. Podwójna pętla `for` tworzy kolejne pola szachownicy od lewej do prawej i z dołu w górę. Gdy suma aktualnej kolumny i wiersza jest parzysta kwadrat jest żółty, a w przeciwnym wypadku czerwony. Następnie aktualna macierz jest zdejmowana ze stosu i układ współrzędnych powraca do poprzedniego stanu.

III. Bufor szablonowy (stencil buffer)

Bufor szablonowy (stencil buffer) jest pomocniczym buforem o rozmiarze takim samym jak bufor kolorów (jak wielkość obszaru OpenGL okna) oraz w zależności od implementacji głębokości od 1 do 8 bitów. Jego zawartość nie jest widoczna na ekranie lecz może posłużyć do wykonywania na niej różnych operacji logicznych, których rezultat określa czy dany fragment przejdzie lub nie test szablonowy.

Konieczność wykorzystania bufora szablonowego należy zgłosić już przy tworzeniu okna OpenGL, a konkretnie przy wybieraniu formatu pikseli. W tym celu w metodzie `bool GL_UstalFormatPikseli()` klasy `TGLForm` w strukturze `opisFormatuPikseli` należy wypełnić dodatkowe pole `cStencilBits` podając żądaną głębokość bufora. W przykładzie z lustrem wystarczy 1 bit co skutkuje dodaniem następującej linii kodu:

```
opisFormatuPikseli.cStencilBits = 1;
```

Pomimo, że żąda się od funkcji `ChoosePixelFormat()` aby wybrała format pikseli z 1-bitowym buforem szablonu może ona zwrócić format pikseli najbliższy podanemu obsługiwany przez sterownik, w którym stencil buffer może mieć np. 8-bitów głębokości. Nie będzie miało to jednak żadnego wpływu na działanie efektu odbić.

Przed użyciem bufora szablonowego należy go wyczyścić podobnie jak bufor kolorów czy głębokości:

```
glClear(GL_STENCIL_BUFFER_BIT);
```

Domyślnie czyszczenie powoduje wypełnienie bufora zerami. Standardową wartość można zmienić podając własną liczbę całkowitą jako parametr funkcji `glClearStencil(GLint s)`. Włączenie testu szablonowego wykonuje instrukcja `glEnable(GL_STENCIL_TEST)`, a wyłączenie analogicznie `glDisable(GL_STENCIL_TEST)`. Do kontrolowania zachowania tego testu służą dwie funkcje. Pierwszą z nich jest:

```
glStencilFunc(GLenum func, Glint ref, Gluint mask)
```

Jej parametry ustawiają:

- funkcję porównującą (`func`),
- wartość referencyjną (`ref`),
- maskę binarną nakładaną operacją `AND` na wartość w szablonie i wartość referencyjną (`mask`).

Stałe określające funkcje porównujące zawiera poniższa tabela.

Wartość	**Funkcja porównująca testu szablonu**
`GL_NEVER`	Wynik testu zawsze będzie negatywny
`GL_ALWAYS`	Wynik testu zawsze będzie pozytywny
`GL_LESS`	Wartość referencyjna < wartość szablonu
`GL_LEQUAL`	Wartość referencyjna <= wartość szablonu
`GL_EQUAL`	Wartość referencyjna == wartość szablonu
`GL_GEQUAL`	Wartość referencyjna >= wartość szablonu
`GL_GREATER`	Wartość referencyjna > wartość szablonu
`GL_NOTEQUAL`	Wartość referencyjna != wartość szablonu

Tabela 1. Funkcje porównujące dla testu szablonu

Sposób wypełniania bufora można określić funkcją:

```
void glStencilOp(GLenum fail, Glenum zfail, Glenum zpass)
```

Jej parametry określają w jaki sposób mają się zmieniać wartości bufora szablonowego gdy:

- test szablonowy się nie powiedzie (`fail`),
- test szablonowy się powiedzie, a test głębokości nie (`zfail`)
- test szablonowy i test głębokości dadzą wynik pozytywny (`zpass`).

Stałe określające możliwe operacje na szablonie zawiera Tabela 2.

Wartość	**Operacja na szablonie**
`GL_KEEP`	Zachowuje aktualną wartość
`GL_ZERO`	Ustawia wartość na zero
`GL_REPLACE`	Zastępuje aktualną wartość wartością referencyjną określoną w funkcji `glStencilFunc()`
`GL_INCR`	Inkrementuje aktualną wartość
`GL_DECR`	Dekrementuje aktualną wartość
`GL_INVERT`	Odwraca bitowo aktualną wartość
`GL_INCR_WRAP`	Inkrementuje aktualną wartość i w razie przekroczenia maksymalnej wartości przyjmuje wartość zero (tylko w OpenGL 1.4 i nowszym)
`GL_DECR_WRAP`	Dekrementuje aktualną wartość i w razie przekroczenia wartości zero przyjmuje wartość maksymalną (tylko w OpenGL 1.4 i nowszym)

Tabela 2. Operacje dokonywane na buforze szablonu

IV. Płaszczyzna obcinająca (clip plane)

Dla każdego kontekstu renderingu OpenGL zdefiniowana jest frusta czyli ścięty ostrosłup, którego wnętrze ogranicza zakres rysowanej sceny. Składa się ona z 6 płaszczyzn, które wyznaczają jej bryłę. Biblioteka OpenGL umożliwia zdefiniowanie dodatkowych własnych płaszczyzn służących do przycinania rysowanych obiektów niezależnie od ustawień kamery. Ich maksymalna ilość określona jest przez daną implementacje i zazwyczaj wynosi 6. Dopuszczalne maksimum zwraca wywołanie funkcji `glGetIntegerv(GL_MAX_CLIP_PLANES)`.

Płaszczyznę można zdefiniować za pomocą jej równania w postaci:

$$A x + B y + C z + D = 0 ,$$

gdzie współczynniki A, B i C można interpretować jako współrzędne wektora normalnego $[A, B, C]$ tej płaszczyzny. Do określenia płaszczyzny obcinającej w OpenGL służy funkcja:

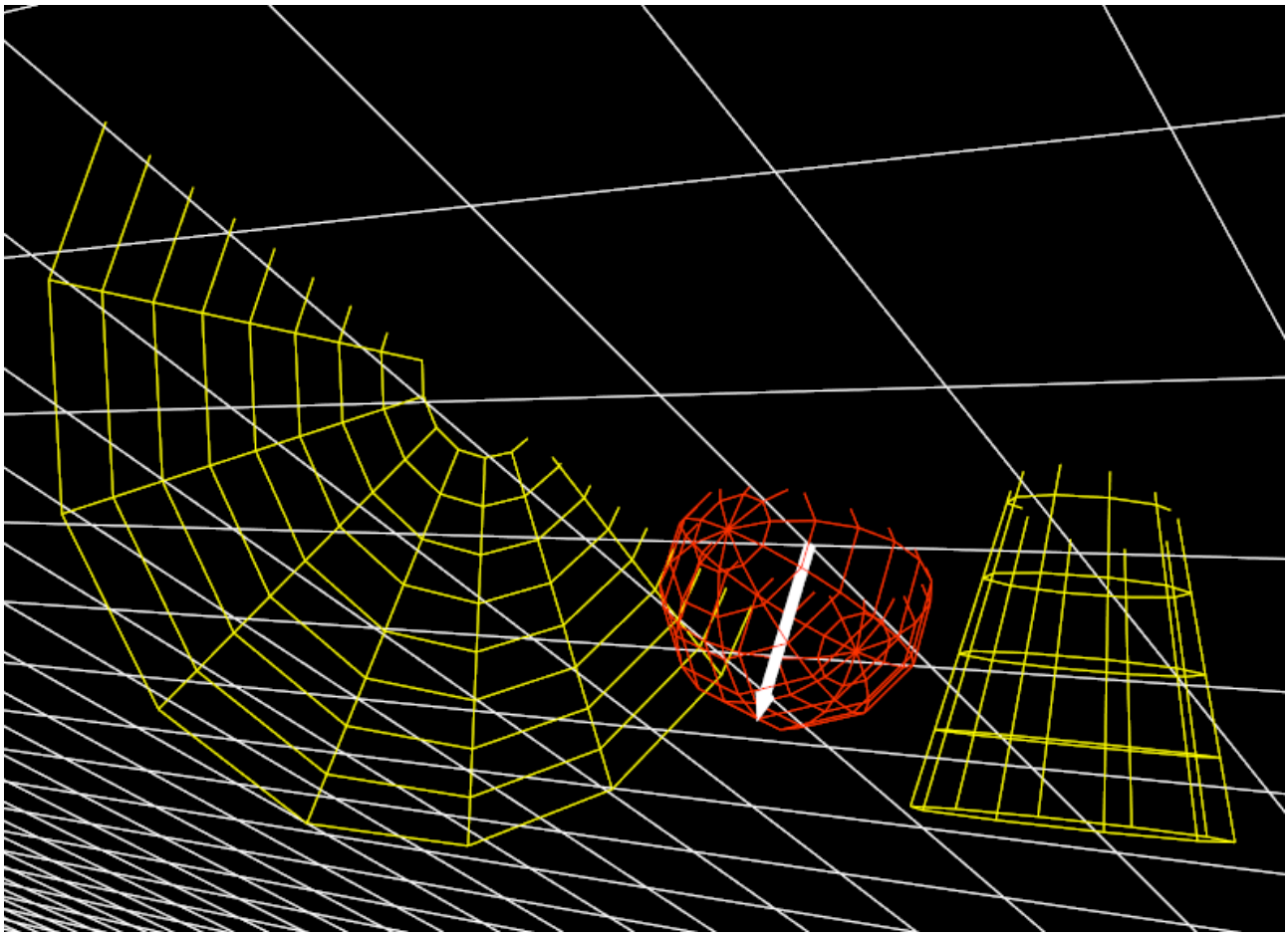
```
void glClipPlane(GLenum plane, const GLdouble *equation)
```

Jej parametry są następujące:

- stała określająca płaszczyznę obcinającą, której równanie jest przekazywane np. `GL_CLIP_PLANE0` (`plane`),
- wskaźnik do tablicy 4 współczynników równania (`equation`).

Biblioteka OpenGL pozwoli na rysowanie obiektów znajdujących się nad daną płaszczyzną czyli po tej połowie przestrzeni, w której stronę jest skierowany wektor normalny. Włączeniem i wyłączeniem przycinania można sterować za pomocą funkcji `glEnable()` i `glDisable()` z parametrem `GL_CLIP_PLANEi`, gdzie `i` jest numerem płaszczyzny.

Działanie płaszczyzny obcinającej demonstruje poniższy rysunek (Rys. 1). Biała siatka to powierzchnia, a strzałka jest jej wektorem normalnym. Przestrzeń podzielona jest na dwie połówki, z których w jednej odbywa się rysowanie, a druga pozostaje nie zmieniona (pusta).



Rys. 1. Działanie płaszczyzny obcinającej.

V. Rysowanie lustra

Funkcja `RysujLustro()` rysująca scenę z lustrem wykonuje swoje zadanie w 3 etapach.

1. Rysowanie szablonu lustra.
2. Rysowanie odbicia.
3. Rysowanie rzeczywistej sceny.

Poniżej opisane są dokładnie poszczególne fazy renderingu.

1. Pierwszy etap polega na narysowaniu w buforze szablonowym kształtu szyby. Będzie on potrzebny w kolejnym punkcie podczas rysowania odbicia. Szablon ten posłuży do przycięcia refleksu do kształtu szyby.
   Rysowanie ma się odbywać tylko w obrębie bufora szablonowego i dlatego należy zablokować bufor kolorów oraz bufor głębokości za co odpowiadają poniższe linie kodu:

```
glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
glDisable(GL_DEPTH_TEST)
```

   Następnie należy włączyć test szablonu i wyczyścić stencil buffer domyślną wartością zero. Kolejną czynnością jest ustawienie jego działania tak aby wynik testu szablonu był zawsze pozytywny oraz każdy punkt, który przejdzie test szablonowy i głębokości (co ma miejsce za każdym razem) zastępował wartość w szablonie wartością referencyjną ustawioną na `1`:

```
glEnable(GL_STENCIL_TEST);
glClear(GL_STENCIL_BUFFER_BIT);
glStencilFunc(GL_ALWAYS, 1, 1);
glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);
```

Teraz można narysować kształt szyby. Ponieważ z założenia obiekty odbijane maja się znajdować tylko po jednej stronie lustra narysowany szablon musi być jednostronny. Z tego powodu należy włączyć ukrywanie tylnych ścianek trójkątów (culling). Ostatecznie kod rysujący szablon wygląda następująco:

```
glEnable(GL_CULL_FACE);
RysujSzybe();
glDisable(GL_CULL_FACE);
```

Zawartość bufora szablonowego po pierwszej fazie rysowania przedstawia Rys. 2.

2. Drugi etap polega na narysowaniu odbicia wewnątrz lustra. W tym celu należy ponownie włączyć bufor kolorów oraz bufor głębokości:

```
glEnable(GL_DEPTH_TEST);
glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
```

Zachowanie buforu szablonowego też musi ulec zmianie. Test szablonowy będzie pozytywny tylko wtedy kiedy wartość w jego buforze będzie równa wartości referencyjnej `1`. Oznacza to, że zostaną narysowane tylko te fragmenty obiektów, które po rzutowaniu na ekran pokrywają się z wcześniej wygenerowanym szablonem szyby. Należy również zablokować dokonywanie wszelkich zmian w buforze szablonowym.

```
glStencilFunc(GL_EQUAL, 1, 1);
glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);
```

Niepożądanym efektem byłoby wystawanie z powierzchni lustra odbitych obiektów dlatego należy je przyciąć. W tym celu należy utworzyć płaszczyznę obcinającą, której równanie wyznaczają współczynniki zawarte w tablicy:

```
const GLdouble rownanie[] = {0.0f, -1.0f, 0.0f, 0.0f};
```

Włączenie przycinania i przekazanie powyższych wartości realizuje się następująco:

```
glEnable(GL_CLIP_PLANE0);
glClipPlane(GL_CLIP_PLANE0, rownanie);
```

W ten sposób rysowanie będzie się odbywać tylko i wyłącznie poniżej płaszczyzny XZ.
W omawianym przykładzie układ współrzędnych odbitych obiektów ma odwrócony kierunek osi Y. Należy więc dokonać odpowiedniego przekształcenia, które jest równoważne przeskalowaniu współrzędnej *y* przez współczynnik *-1*. Aby nie utracić poprzednich przekształceń konieczne jest również zachowanie aktualnej macierzy model – widok poprzez wrzucenie jej na stos. Następnie należy narysować odbijane obiekty. Drugi etap ostatecznie kończy zdjęcie poprzedniej macierzy ze stosu. Całość rysowania odbicia przedstawia poniższy kod:

```
glPushMatrix();
   glScalef(1.0f, -1.0f, 1.0f);
   RysujScene();
glPopMatrix();
```

W dalszej części funkcji nie będzie już potrzebna powierzchnia obcinająca oraz bufor szablonowy dlatego należy je wyłączyć:

```
glDisable(GL_CLIP_PLANE0);
glDisable(GL_STENCIL_TEST);
```

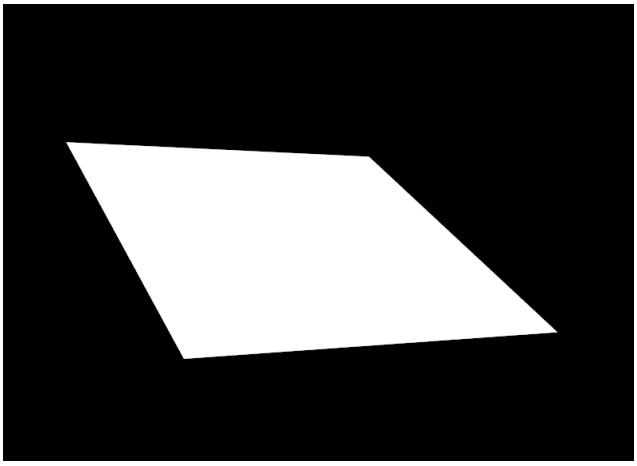
Wynik działania etapu drugiego można zobaczyć na Rys. 3.

3. Ostatnia faza ma za zadanie wygenerować rzeczywistą scenę. Dlatego konieczne jest kolejne wywołanie funkcji rysującej wszystkie obiekty lecz tym razem w przestrzeni bez odwróconej współrzędnej *y*. Ponadto na sam koniec należy narysować półprzeźroczystą szybę wykorzystując mieszanie kolorów (blending). Ostatni etap realizuje poniższy kod:

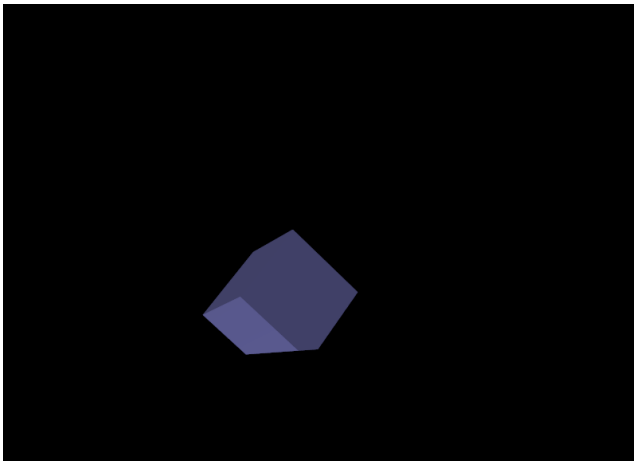
```
   RysujScene();
   glEnable(GL_BLEND);
   RysujSzybe();
   glDisable(GL_BLEND);
```

Ostateczny rezultat działania funkcji `RysujLustro()` przedstawia Rys. 4. Jej pełny kod wygląda następująco:

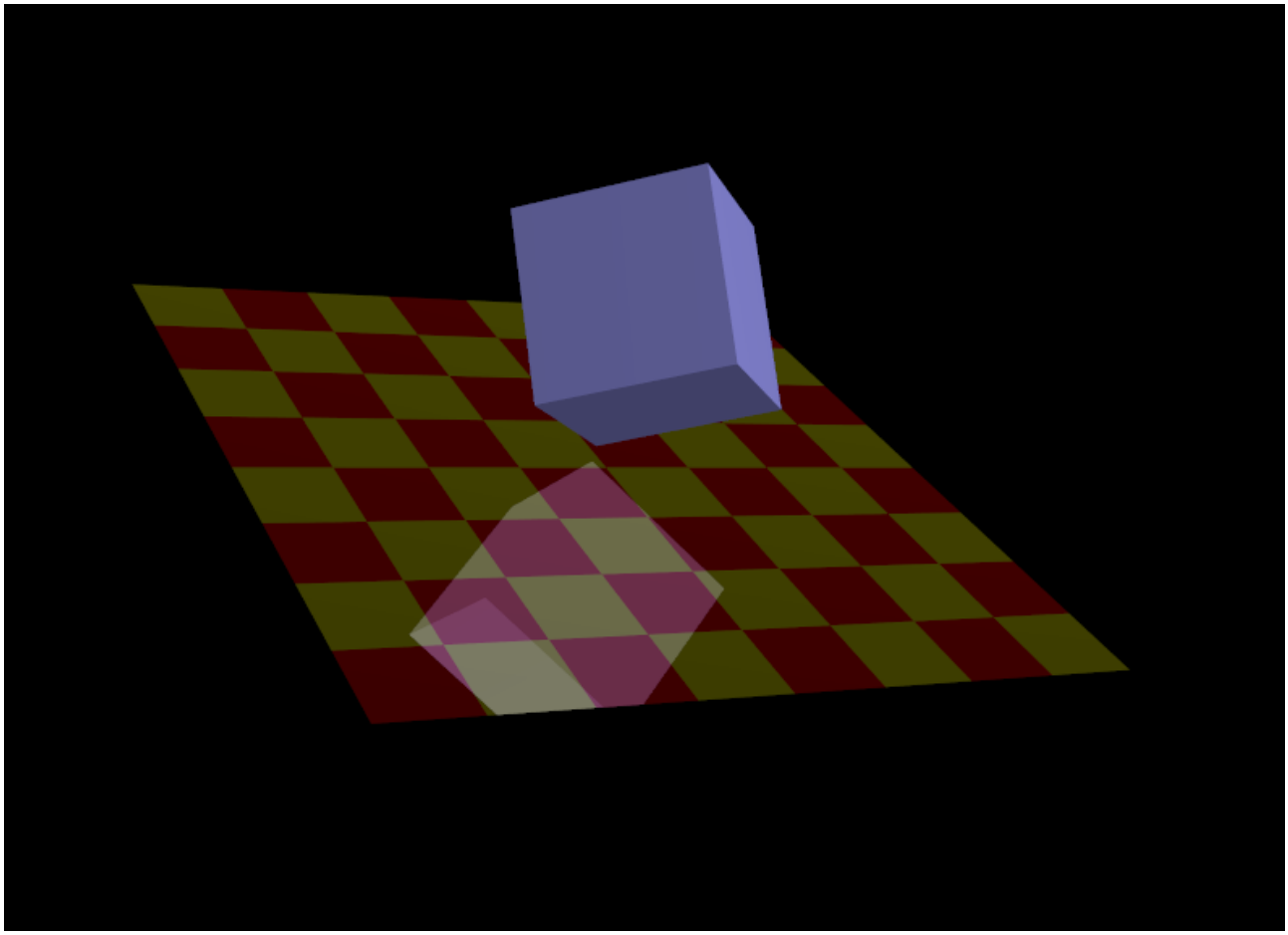
```
void __fastcall TForm1::RysujLustro()
{
      const GLdouble rownanie[] = {0.0f, -1.0f, 0.0f, 0.0f};

      /* 1. Rysowanie szablonu lustra */
      //wyłączenie rysowania w buforze kolorów
      glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
      glDisable(GL_DEPTH_TEST); //wyłaczenie testu głebokości

      glEnable(GL_STENCIL_TEST); //właczenie testu szablonu
      glClear(GL_STENCIL_BUFFER_BIT); //czyszczenie bufora szablonu
      glStencilFunc(GL_ALWAYS, 1, 1);
      glStencilOp(GL_KEEP, GL_KEEP, GL_REPLACE);

      //rysowanie kształtu szyby
      glEnable(GL_CULL_FACE);
      RysujSzybe();
      glDisable(GL_CULL_FACE);

      /* 2. rysowanie odbicia w obrębie szablonu */
      glEnable(GL_DEPTH_TEST);
      glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);

      glStencilFunc(GL_EQUAL, 1, 1);
      glStencilOp(GL_KEEP, GL_KEEP, GL_KEEP);

      glEnable(GL_CLIP_PLANE0);
      glClipPlane(GL_CLIP_PLANE0, rownanie);
      glPushMatrix();
            glScalef(1.0f, -1.0f, 1.0f); //odbicie lustrzane osi Y
            RysujScene();
      glPopMatrix();

      //wyłaczenie powierzchni obcinającej i bufora szablonu
      glDisable(GL_CLIP_PLANE0);
      glDisable(GL_STENCIL_TEST);

      /* 3. Rysowanie rzeczywistej sceny */
      RysujScene();

      //rysowanie półprzeźroczystej szyby
      glEnable(GL_BLEND);
      RysujSzybe();
      glDisable(GL_BLEND);
}
```



Rys. 2. Wynik działania pierwszego etapu rysowania – wizualizacja zawartości bufora szablonu. Kolor czarny odpowiada wartości 0, a kolor biały wartości 1.



Rys. 3. Zawartość bufora kolorów po drugim etapie rysowania. Odbicie jest przycięte do kształtu szablonu z etapu pierwszego.



Rys. 4. Ostateczna scena otrzymana w trzeciej fazie renderowania. Na rezultat poprzednich etapów rysowania zostały nałożone obiekty rzeczywistej sceny oraz półprzezroczysta szyba.

VI. Uwagi

W przypadku zastosowania opisanego algorytmu można się spotkać z pewnymi problemami. Jeden z nich może wystąpić podczas używania świateł podczas rysowania sceny. Fragmenty renderingu, które są odbiciami mogą zostać oświetlone w taki sam sposób jak obiekty rzeczywistej sceny bez uwzględnienia odwrócenia układu współrzędnych. Może to być spowodowane tym, że pozycje świateł zostały określone przed skalowaniem macierzy model – widok. Aby temu zapobiec należy ponownie ustawić światła przed rysowaniem odbić oraz przed rysowaniem rzeczywistej sceny.

Kolejny problem pojawia się w momencie gdy geometria jest bardzo złożona i rendering długotrwały. Jedną z możliwych w takim wypadku optymalizacji jest zmniejszenie szczegółowości odbicia poprzez np. rysowanie mniejszej ilości obiektów wewnątrz lustra.