Modelowanie i Analiza Sytemów Grafiki Komputerowej

Programowy potok renderingu

Rasteryzacja trójkątów

Dominik Szajerman

Instytut Informatyki
Politechnika Łódzka

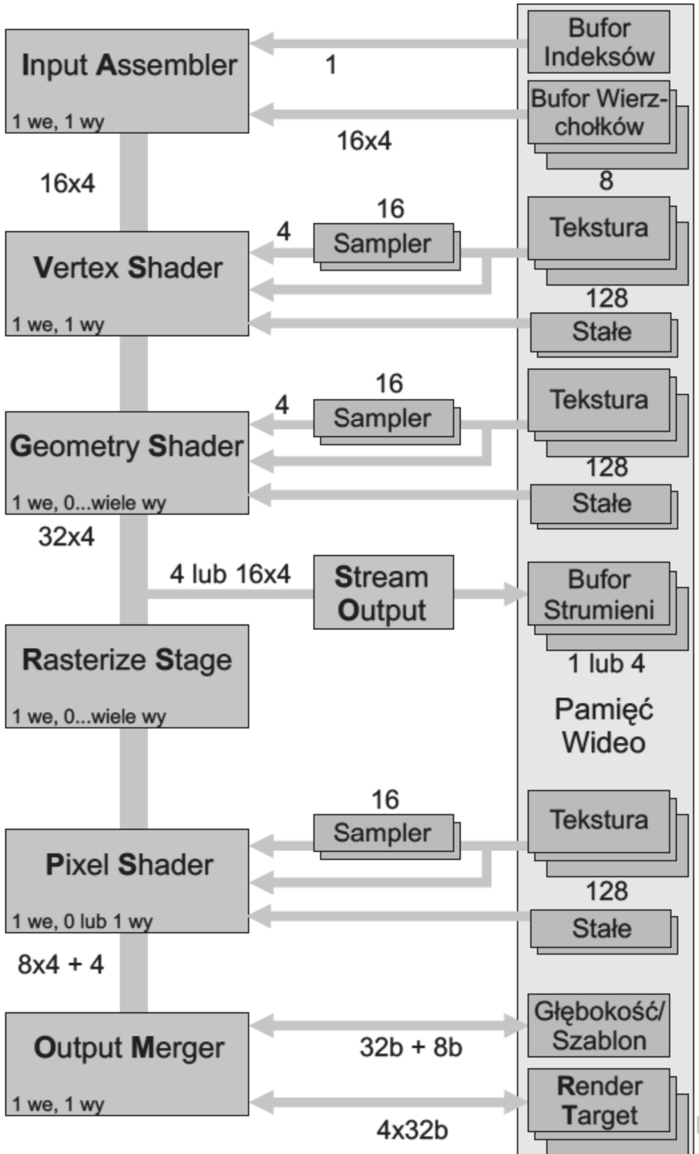
2013/2014

Potok renderingu

Proces przetworzenia opisu sceny trójwymiarowej danego pewną reprezentacją na dwuwymiarowy obraz rastrowy:

- klasyczny (fixed-function) - wsparcie OpenGL i DirectX,
- programowalny - programy cieniowania na procesorze graficznym.

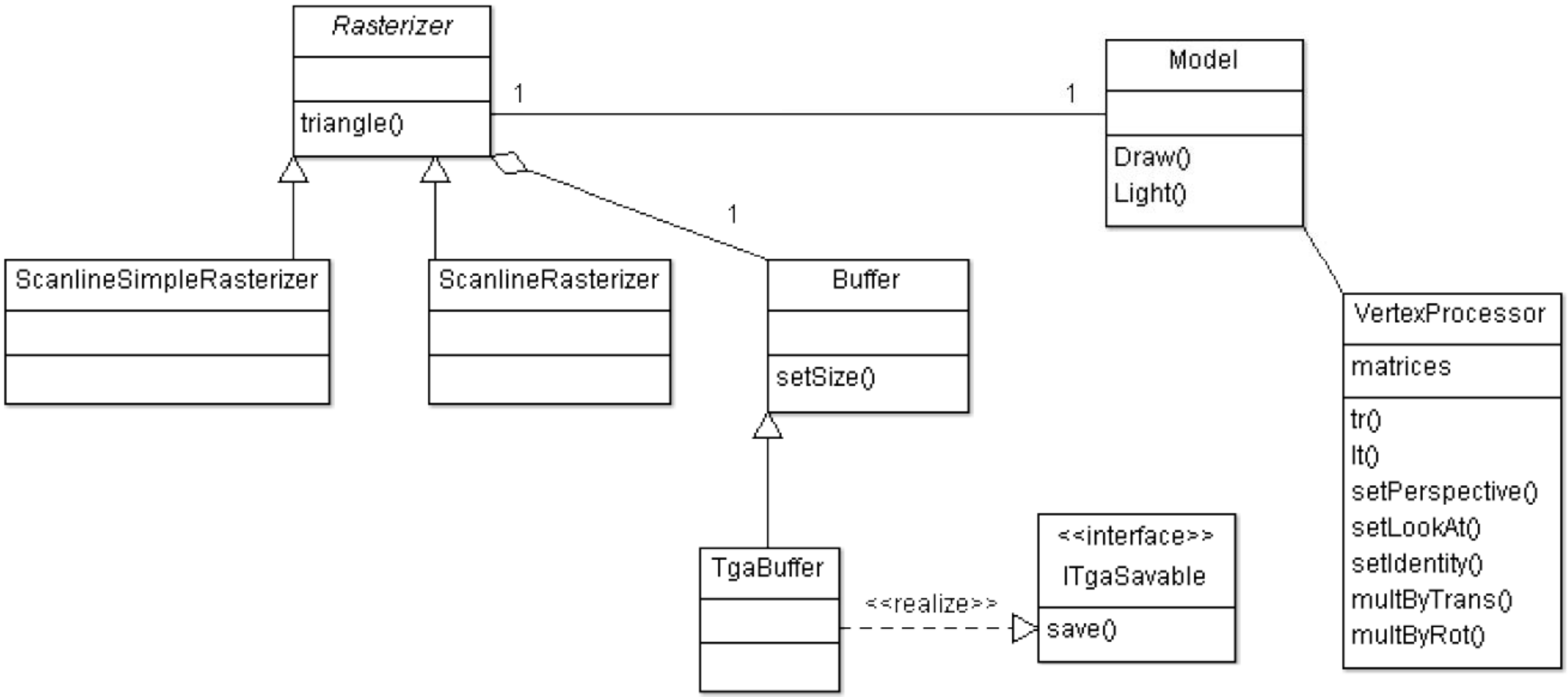
Shader Model 4.0



Programowy potok renderingu

1. bufor renderingu
2. rasteryzacja
   - culling
   - obcinanie
   - interpolacja
   - konwencja wypełniania
   - z-bufor
3. transformacje
4. oświetlenie

Architektura aplikacji



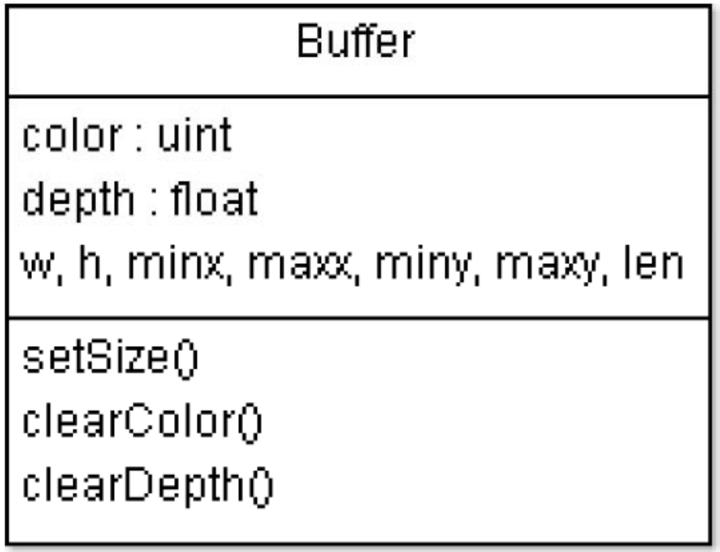
Typy danych

Dodatkowe typy danych dla wsparcia obliczeń renderingu:

1. wektory trój- i czterowymiarowe - długość, normalizacja, iloczyn skalarny, iloczyn wektorowy (3D), mnożenie elementami, arytmetyczne, odbicie,
2. macierze 4×4 - mnożenie przez wektor i przez macierz.

Nazwy według konwencji znanej z shaderów oraz CUDA: float3, float4, float4x4...

Bufor Koloru



$\rightarrowtail$

Format TGA - zapis

```
// wariant niekompresowany, 32 bity na piksel
unsigned short header[9]={
  0x0000, 0x0002, 0x0000, 0x0000, 0x0000, 0x0000,
  0x0100, 0x0100, // width, height
  0x0820};

FILE *f = fopen( "nazwapliku.tga", "wb+" );
if (NULL == f) return -1;

header[6] = width;
header[7] = height;

fwrite( header, 2, 9, f );
fwrite( colorBuffer, 4, width*height, f );

fclose(f);
```

Rasteryzacja

Działanie polegające na odwzorowaniu kształtu na medium o skończonej rozdzielczości.

W renderingu 3D potrzebna jest rasteryzacja trójkątów.

Jest operacją znaną, ale słabo udokumentowaną - obecnie wszystko to robią karty graficzne i nie trzeba wiedzieć, jak to działa.

Rasteryzacja

Działanie polegające na odwzorowaniu kształtu na medium o skończonej rozdzielczości.

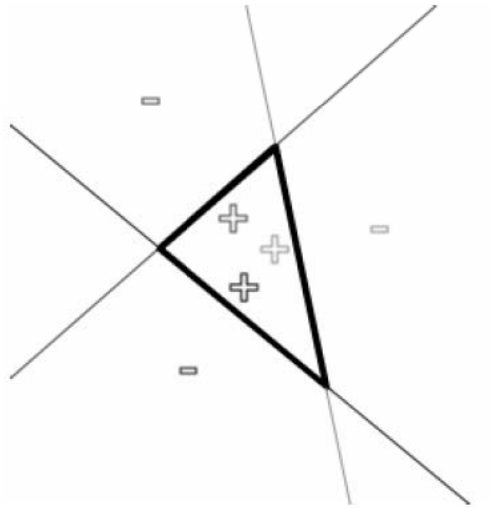
W renderingu 3D potrzebna jest rasteryzacja trójkątów.

Jest operacją znaną, ale słabo udokumentowaną - obecnie wszystko to robią karty graficzne i nie trzeba wiedzieć, jak to działa.

Ale warto ;o)
„Knowledge of these algorithm simply make you a better graphics programmer."

Funkcja „half-space”

- przyjmuje wartości dodatnie z jednej strony prostej a ujemne z jej drugiej strony - czyli dzieli płaszczyznę na pół
- jest równa zero na prostej
- każda krawędź trójkąta dzieli płaszczyznę



- tam, gdzie wszystkie na raz są dodatnie, tam jest wnętrze trójkąta
- znając równanie takiej funkcji można wyznaczać, które piksele znajdują się we wnętrzu trójkąta

Równanie prostej

Równanie prostej przechodzącej przez dwa punkty (x_1, y_1) i (x_2, y_2):

$$(x_2 - x_1) \cdot (y - y_1) - (y_2 - y_1) \cdot (x - x_1) = 0 \tag{1}$$

Równanie prostej

Równanie prostej przechodzącej przez dwa punkty (x_1, y_1) i (x_2, y_2):

$$(x_2 - x_1) \cdot (y - y_1) - (y_2 - y_1) \cdot (x - x_1) = 0 \tag{1}$$

Funkcja

$$f(x, y) = (x_2 - x_1) \cdot (y - y_1) - (y_2 - y_1) \cdot (x - x_1) \tag{2}$$

spełnia założenia funkcji „half-space”.

Wyznaczanie wnętrza trójkąta

Dla trójkąta o wierzchołkach $(x_1, y_1), (x_2, y_2), (x_3, y_3)$ ułożonych zgodnie z ruchem zegara, jego wnętrze stanowią wszystkie punkty (x, y) spełniające równanie:

$$
\begin{aligned}
& (x_1 - x_2) \cdot (y - y_1) - (y_1 - y_2) \cdot (x - x_1) > 0 \\
\wedge \quad & (x_2 - x_3) \cdot (y - y_2) - (y_2 - y_3) \cdot (x - x_2) > 0 \\
\wedge \quad & (x_3 - x_1) \cdot (y - y_3) - (y_3 - y_1) \cdot (x - x_3) > 0
\end{aligned}
\tag{3}
$$

Kanoniczna bryła widzenia

Rzutowanie przekształca przestrzeń tak, że bryła widzenia dana sześcioma ścianami (np. `glFrustum`, `glOrtho`, `gluPerspective`) staje się sześcianem o współrzędnych $[-1,-1,-1]-[1,1,1]$ i długości boku $= 2$.

Dlatego współrzędne (x, y, z) widocznych wierzchołków wchodzące do funkcji rasteryzującej trójkąt mieszczą się w takim zakresie a pozostałe powinny być obcinane.

Aby przeształcić współrzędne kanoniczne pikseli na współrzędne w oknie renderingu (zakres $0 \ldots \text{width} \times 0 \ldots \text{height}$) należy zastosować poniższe wzory.

$$\begin{aligned} x' &= (x+1) * \text{width} * .5f \\ y' &= (y+1) * \text{height} * .5f \end{aligned} \tag{4}$$

Optymalizacja 1 - przeszukiwanie

Testowi należy poddać tylko piksele w prostokącie zawierającym trójkąt. Taki prostokąt łatwo wyznaczyć.

```
minx = min(x1, x2, x3)
maxx = max(x1, x2, x3)
miny = min(y1, y2, y3)
maxy = max(y1, y2, y3)
```

Culling

Culling - odrzucanie ścianek, które są tyłem. Zgodnie z powyższą konwencją tyłem ustawione są ścianki rysowane przeciwnie do ruchu wskazówek zegara. $\rightarrowtail$

Obcinanie

Pomijanie pikseli wchodzących w skład trójkąta, ale nie mieszczących się w buforze wyjściowym.

Należy to zrobić w metodzie rasteryzującej ponieważ dopiero tam znane są potencjalne współrzędne pikseli wchodzących w skład trójkąta.

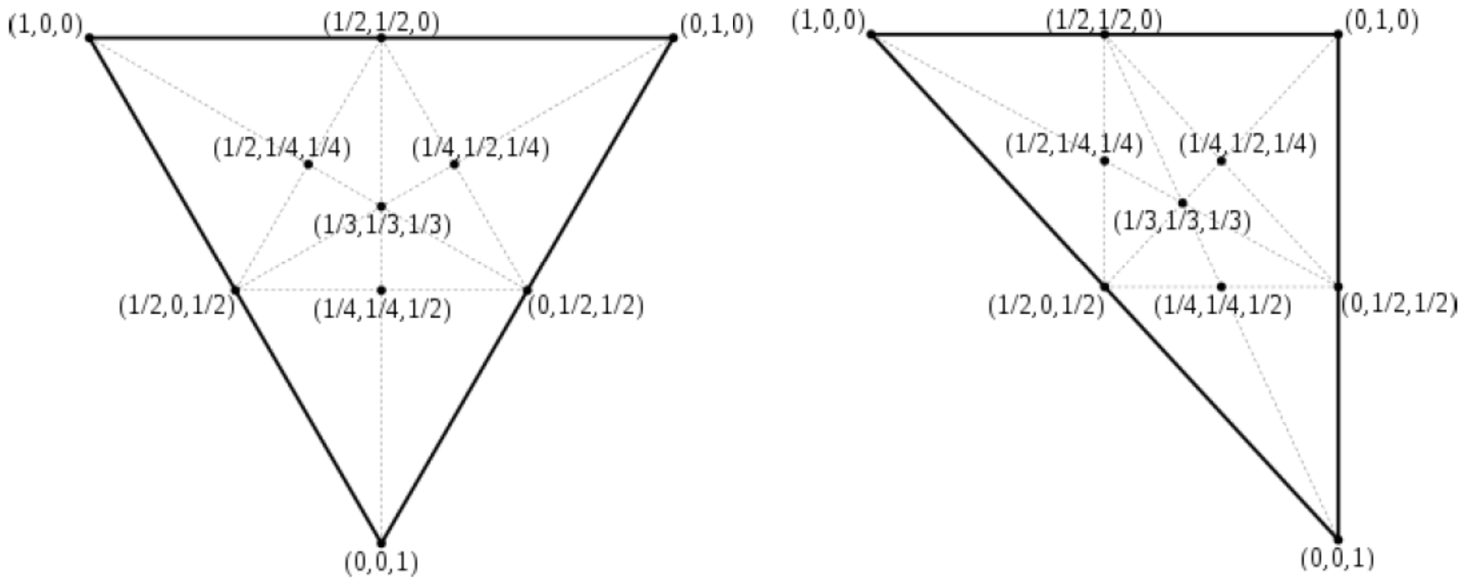
```
minx = max(minx, 0);
maxx = min(maxx, width-1);
miny = max(miny, 0);
maxy = min(maxy, height-1);
```

Optymalizacja 2 - stałe

```
float dx12 = x1-x2;
float dx23 = x2-x3;
float dx31 = x3-x1;
float dy12 = y1-y2;
float dy23 = y2-y3;
float dy31 = y3-y1;
```

Interpolacja - współrzędne barycentryczne

Współrzędne barycentryczne - układ współrzędnych zdefiniowany przez wierzchołki trójkąta.



$$\lambda_1 + \lambda_2 + \lambda_3 = 1 \tag{5}$$

$$\begin{aligned}\lambda_1 &= \frac{(y_2-y_3)(x-x_3)+(x_3-x_2)(y-y_3)}{(y_2-y_3)(x_1-x_3)+(x_3-x_2)(y_1-y_3)}\\ \lambda_2 &= \frac{(y_3-y_1)(x-x_3)+(x_1-x_3)(y-y_3)}{(y_3-y_1)(x_2-x_3)+(x_1-x_3)(y_2-y_3)}\\ \lambda_3 &= 1-\lambda_1-\lambda_2\end{aligned} \tag{6}$$

Interpolacja

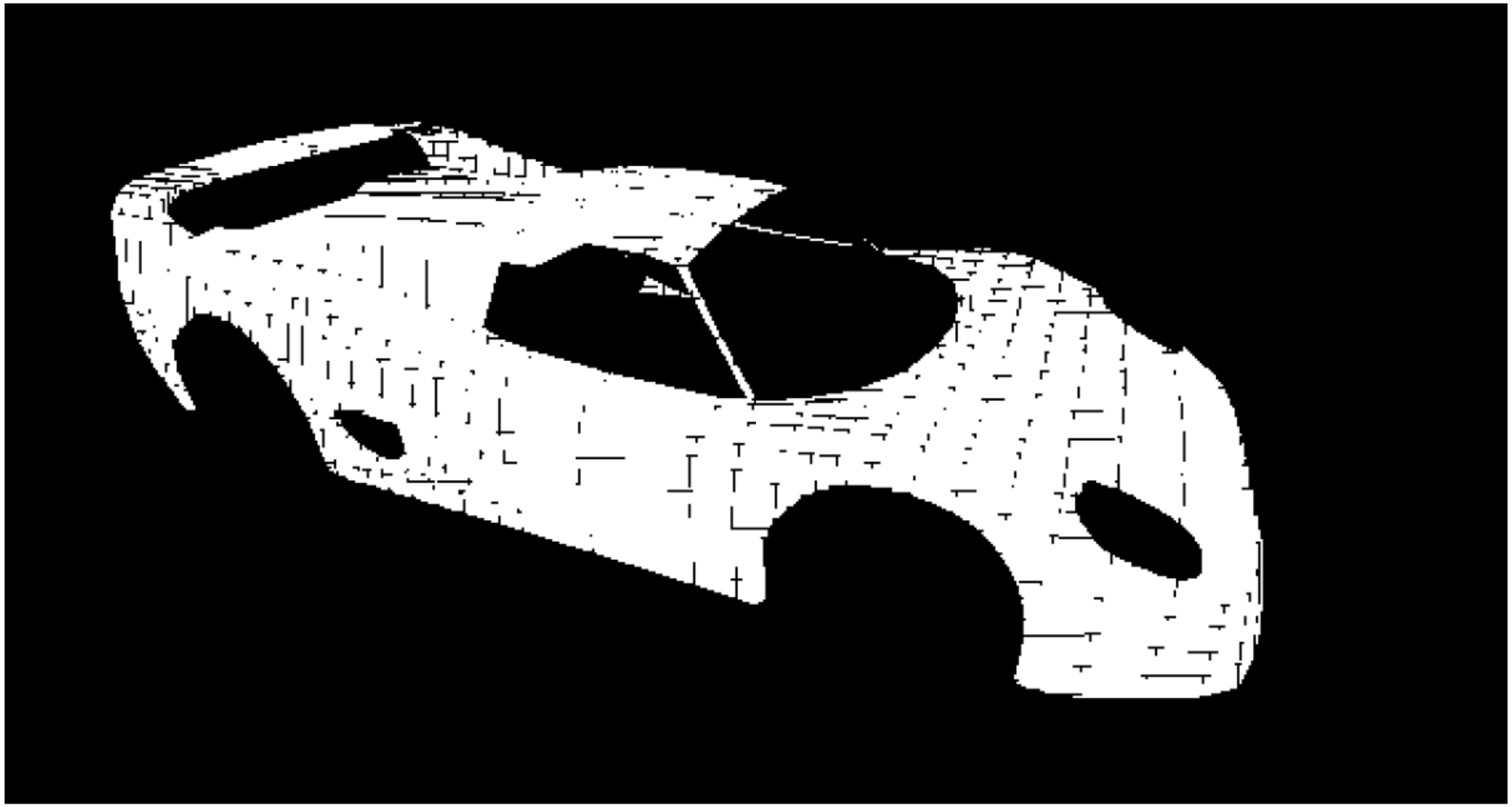
Interpolacja np. koloru w wierzchołkach $(c1, c2, c3)$:

$$c = \lambda_1 \cdot c_1 + \lambda_2 \cdot c_2 + \lambda_3 \cdot c_3 \tag{7}$$

$\rightarrowtail$

Konwencja wypełniania 1

Nierówności ostre w równaniu 3 powodują, że nie są renderowane piksele leżące na brzegu trójkąta. Powoduje to powstawanie przerw między renderowanymi trójkątami.

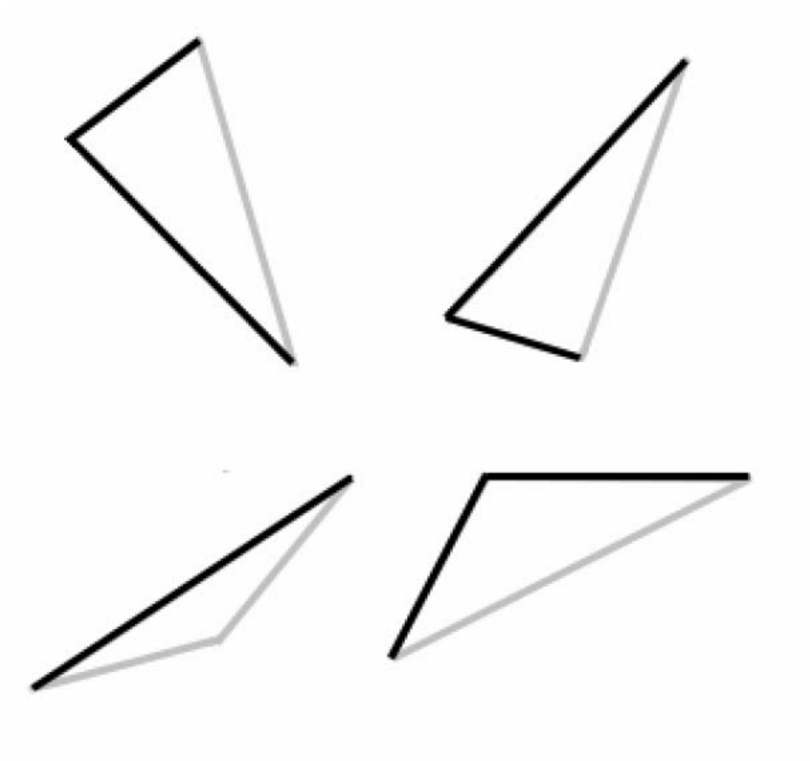


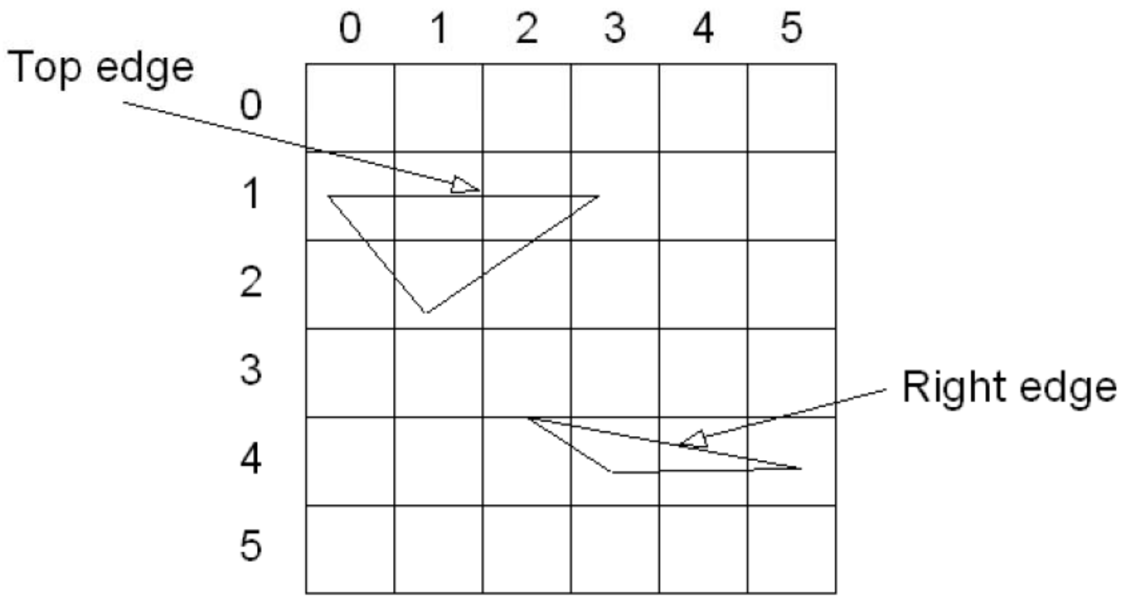
Konwencja wypełniania 2

Z drugiej strony wstawienie tam nierówności nieostrych spowoduje wielokrotne renderowanie pokrywających się pikseli, a co za tym idzie artefakty renderingu.

Aby rozwiązać ten problem biblioteki graficzne OpenGL, DirectX i GDI korzystają z *top-left filling convection*. Polega to na tym, że „górne" i "lewe" krawędzie danego trójkąta renderowane są przy pomocy nierówności nieostrych (czyli brzeg jest renderowany) a „dolne" i „prawe" przy pomocy ostrych.

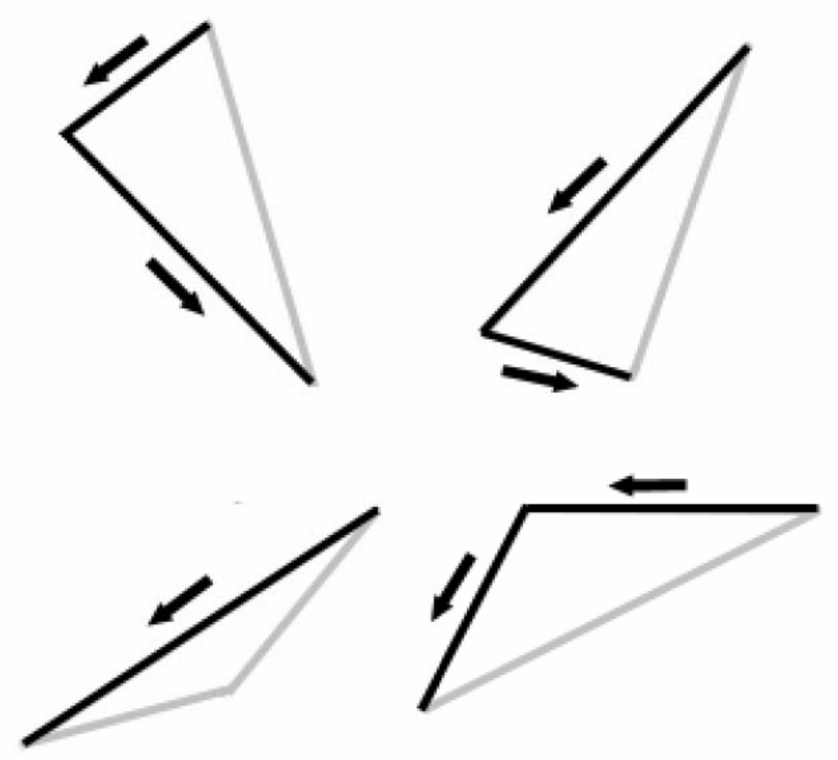
Krawędzie „*top-left*”



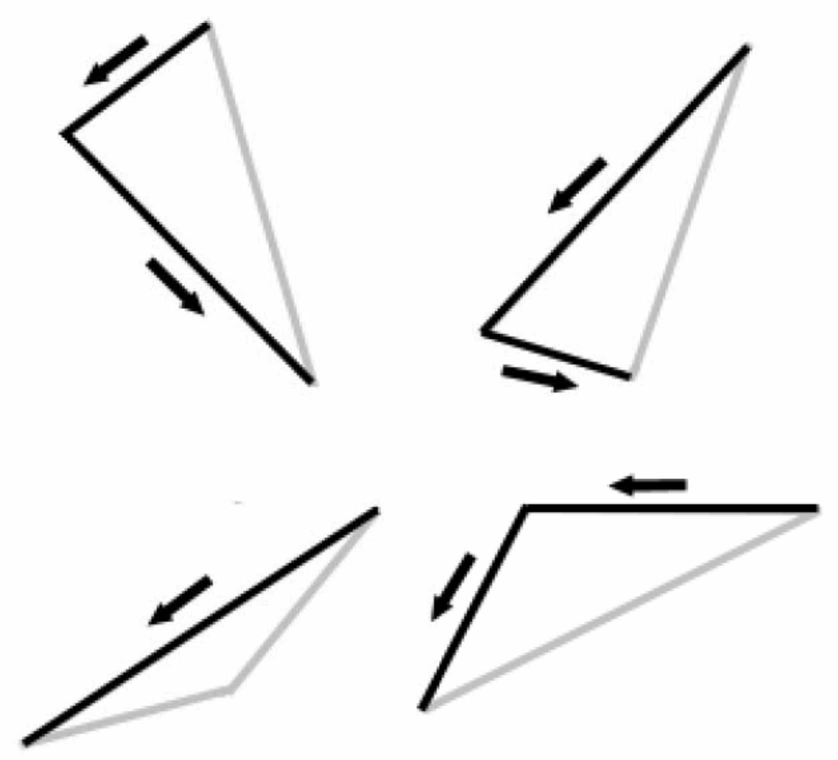


Jak je wykryć (obliczyć)?

Wykrywanie krawędzi „*top-left*”

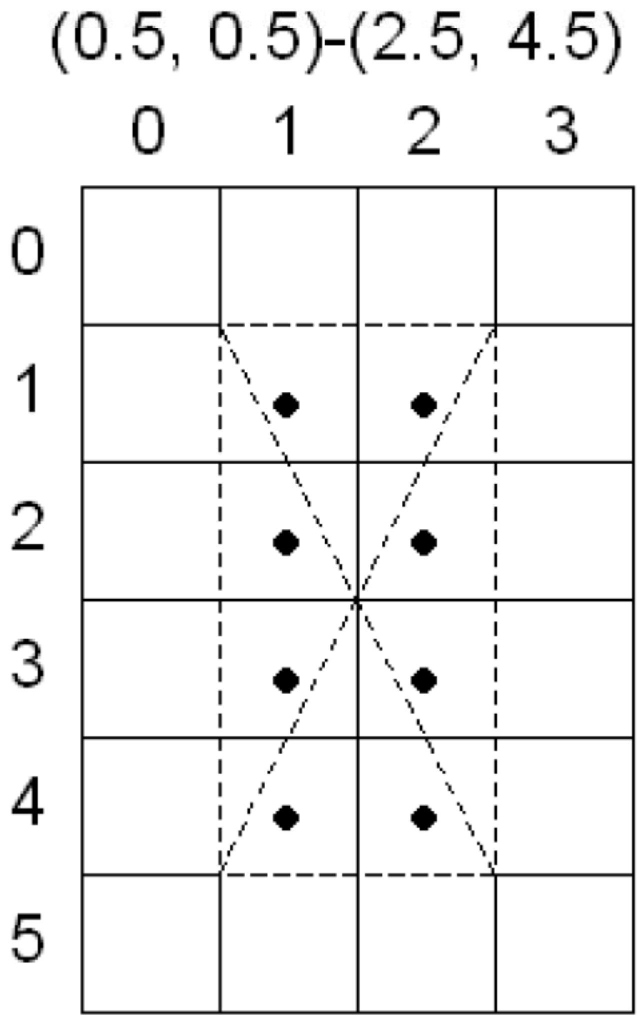


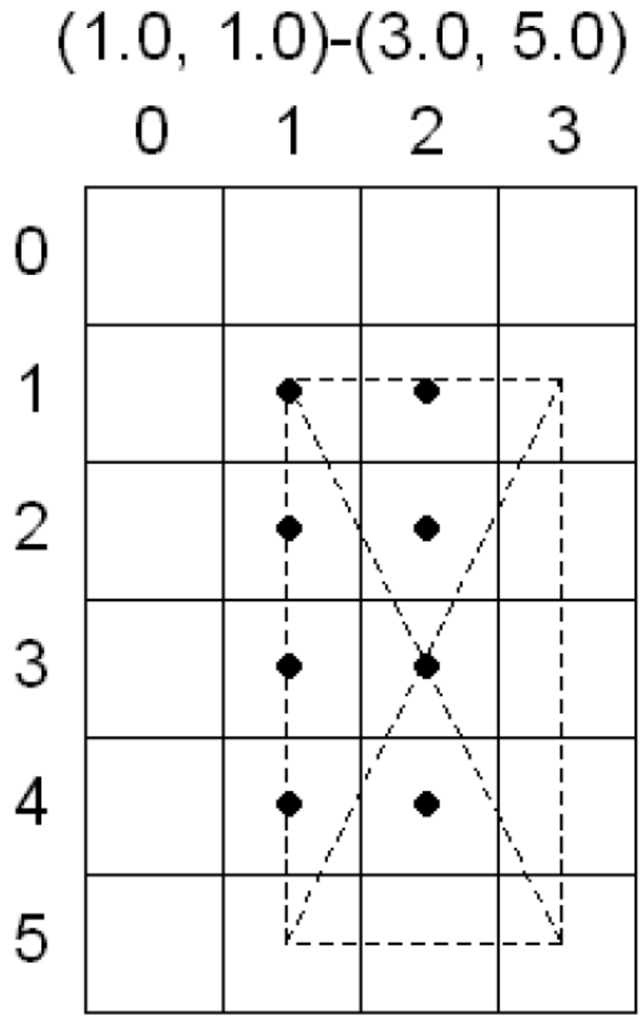
Wykrywanie krawędzi „*top-left*”



```
if (dy12 < 0 || (dy12 == 0 && dx12 > 0)) { tl1=true; }
if (dy23 < 0 || (dy23 == 0 && dx23 > 0)) { tl2=true; }
if (dy31 < 0 || (dy31 == 0 && dx31 > 0)) { tl3=true; }
```

Położenie pikseli po rasteryzacji





Decyduje środek piksela.

Bufor Głębokości

```
float depth
  = ( lambda1 * v1.z + lambda2 * v2.z + lambda3 * v3.z );
if ( depth < depthBuffer[x] )
{
    colorBuffer[x] = color;
    depthBuffer[x] = depth;
}
```

$\rightarrowtail$