Grafika komputerowa

dr inż. Maciej Zakrzewicz
Instytut Informatyki
Politechnika Poznańska

Program przedmiotu

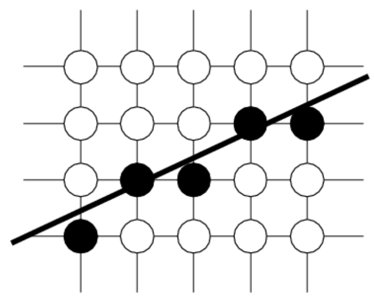
- Grafika 2D
  - algorytmy rastrowe (kreślenie odcinka, okręgu, wypełnianie wielokątów, aproksymacja półtonowa)
  - obcinanie 2D
  - transformacje 2D
- Grafika 3D
  - transformacje 3D
  - rzutowanie 3D->2D
  - wyznaczanie powierzchni widocznych (algorytm ogólny, z-bufor)
  - oświetlenie i cieniowanie (model oświetlenia, cieniowanie Gourauda i Phonga)
  - algorytm śledzenia promieni
- Reprezentowanie krzywych i powierzchni
  - krzywe i powierzchnie Hermite'a, Beziera, B-sklejane

Literatura

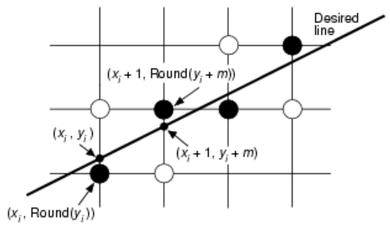
- „Wprowadzenie do grafiki komputerowej", J.D. Foley, A. van Dam, S.K. Feiner, J.F. Hughes, R.L. Phillips, wydanie drugie, WNT 1995, 2001, ISBN 83-204-2662-6
- „Elementy grafiki komputerowej", M. Jankowski, WNT 1990, ISBN 83-204-1326-5

Kreślenie odcinków

- Algorytm kreślenia odcinka oblicza współrzędne pikseli, które leżą na lub blisko idealnej nieskończenie cienkiej linii prostej nałożonej na siatkę dwuwymiarowego rastra
- Złożoność obliczeniowa algorytmu powinna być minimalna; unikanie złożonych wyrażeń arytmetycznych i operacji zmiennoprzecinkowych



Podstawowy algorytm przyrostowy



$$m = \frac{\Delta y}{\Delta x}$$

$$y_i = mx_i + B$$

$$y_{i+1} = mx_{i+1} + B = m(x_i + \Delta x) + B = y_i + m\Delta x$$

$$\Delta x = 1 \Rightarrow y_{i+1} = y_i + m$$

Podstawowy algorytm przyrostowy

```
void Line(int x0, int y0, int x1, int y1, int value)
{               /* Assumes -1 <= m <= 1, x0 < x1 */
  int x;         /* x runs from x0 to x1 in unit increments.*/
  float dy,dx,y,m;

  dy = y1 - y0;
  dx = x1 - x0;
  m = dy/dx;
  y = y0;
  for( x=x0; x<=x1; x++ ){
     WritePixel(x,(int)floor(y+0.5),value); /* Set pixel to value */
     y+=m;                              /* Step y by slope m */
  }
}
```

Algorytm z punktem środkowym

Równanie odcinka:

$$F(x,y) = ax + by + c = 0$$

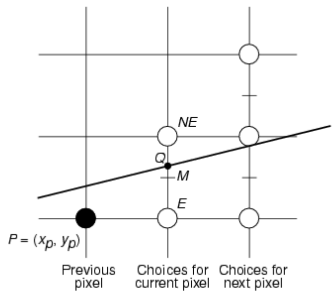
$$dy = y_1 - y_0,\ dx = x_1 - x_0$$

$$y = \frac{dy}{dx}x + B$$

$$F(x,y) = dy \cdot x - dx \cdot y + B \cdot dx$$

$$a = dy,\ b = -dx,\ c = B \cdot dx$$

F(x,y)=0 dla punktów leżących na odcinku
F(x,y)>0 dla punktów poniżej odcinka
F(x,y)<0 dla punktów powyżej odcinka



Wprowadzamy zmienną decyzyjną:

$$d = F(M) = F(x_p + 1, y_p + 1/2)$$

$$d = a(x_p + 1) + b(y_p + 1/2) + c$$

Jeżeli d>0, to wybieramy NE
Jeżeli d≤0, to wybieramy E

Algorytm z punktem środkowym

Jaka będzie wartość *d* dla następnej linii siatki?

- jeżeli wybrano E, to M jest zwiększane o jeden krok w kierunku *x*

$$d_{old} = a(x_p + 1) + b(y_p + 1/2) + c$$

$$d_{new} = F(x_p + 2, y_p + 1/2) = a(x_p + 2) + b(y_p + 1/2) + c$$

$$d_{new} = d_{old} + a$$

- jeżeli wybrano NE, to M jest zwiększane o jednostkę w obu kierunkach *x* i *y*

$$d_{old} = a(x_p + 1) + b(y_p + 1/2) + c$$

$$d_{new} = F(x_p + 2, y_p + 3/2) = a(x_p + 2) + b(y_p + 3/2) + c$$

$$d_{new} = d_{old} + a + b$$

Algorytm z punktem środkowym

Jaka będzie początkowa wartość *d* ?

$$F\left(x_0+1, y_0+1/2\right)=a(x_0+1)+b(y_0+1/2)+c=$$
$$=ax_0+by_0+c+a+b/2=F(x_0,y_0)+a+b/2$$
$$F(x_0,y_0)=0$$
$$d_{start}=a+b/2$$

W celu uniknięcia ułamka w d_{start} zmieniamy oryginalną funkcję *F* mnożąc ją przez 2:

$$F(x,y)=2(ax+by+c)$$

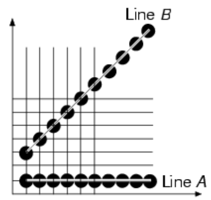
Algorytm z punktem środkowym

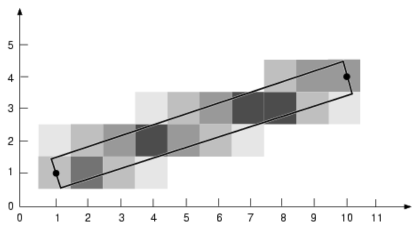
```
void MidpointLine(int x0, int y0, int x1, int y1, int value)
{
  int dx, dy, incrE, incrNE, d, x, y;

  dx = x1 - x0;
  dy = y1 - y0;
  d = dy*2 - dx; /* Initial value of d */
  incrE = dy*2;  /* Increment used for move to E */
  incrNE = (dy-dx)*2;     /* Increment used for move to NE */
  x = x0;
  y = y0;
  WritePixel(x, y, value);         /* The start pixel */
  while(x < x1){
        if(d <= 0){                /* Choose E */
              d += incrE;
              x++;
        }else{                     /* Choose NE */
              d += incrNE;
              x++;
              y++;
        }
        WritePixel(x, y, value); /* The selected pixel closest to the line */
  }
}
```

Kreślenie odcinków – uwagi dodatkowe

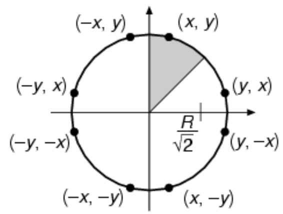
- Zmiana jasności w funkcji nachylenia; konieczne wprowadzenie kompensacji, uzależniając jasność od nachylenia odcinka
- Kreślony odcinek nie posiada zerowej grubości; nie powinien mieć tylko jednego czarnego piksela w kolumnie, lecz powinien raczej wnosić pewien udział do jasności każdego piksela w kolumnie, którego powierzchnię przecina





Kreślenie okręgów – ośmiokrotna symetria

- Do pełnego określenia okręgu wystarczy wykonać obliczenia tylko dla segmentu o kącie 45 stopni (oktantu); pozostałych siedem segmentów może być wyświetlonych symetrycznie



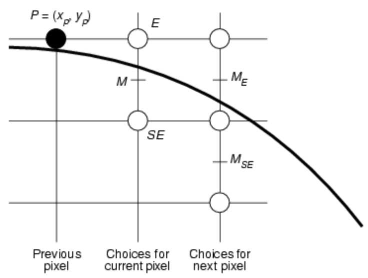

```
void CirclePoints(float x, float y, int value)
{
  WritePixel( x,  y, value);
  WritePixel( y,  x, value);
  WritePixel( y, -x, value);
  WritePixel( x, -y, value);
  WritePixel(-x, -y, value);
  WritePixel(-y, -x, value);
  WritePixel(-y,  x, value);
  WritePixel(-x,  y, value);
}
```

Algorytm z punktem środkowym

Równanie okręgu:

$$F(x,y) = x^2 + y^2 - R^2$$

F(x,y)=0 dla punktów leżących na okręgu
F(x,y)>0 dla punktów na zewnątrz okręgu
F(x,y)<0 dla punktów wewnątrz okręgu



Wprowadzamy zmienną decyzyjną *d*:

$$d = F(x_p+1, y_p-1/2) = (x_p+1)^2 + (y_p-1/2)^2 - R^2$$

Jeżeli d≥0, to wybieramy SE
Jeżeli d<0, to wybieramy E

Algorytm z punktem środkowym

Jaka będzie wartość *d* dla następnej linii siatki?

- jeżeli wybrano E, to M jest zwiększane o jeden krok w kierunku *x*

$$d_{old} = (x_p+1)^2 + (y_p-1/2)^2 - R^2$$

$$d_{new} = F(x_p+2, y_p-1/2) = (x_p+2)^2 + (y_p-1/2)^2 - R^2$$

$$d_{new} = d_{old} + (2x_p+3)$$

- jeżeli wybrano SE, to M jest zwiększane o jednostkę w kierunku *x* i zmniejszane o jednostkę w kierunku *y*

$$d_{old} = (x_p+1)^2 + (y_p-1/2)^2 - R^2$$

$$d_{new} = F(x_p+2, y_p-3/2) = (x_p+2)^2 + (y_p-3/2)^2 - R^2$$

$$d_{new} = d_{old} + (2x_p - 2y_p + 5)$$

Algorytm z punktem środkowym

Jaka będzie początkowa wartość *d*?

$$F(x_0+1, y_0-1/2) = F(1, R-1/2) =$$

$$= 1 + (R^2 - R + 1/4) - R^2 = 5/4 - R$$

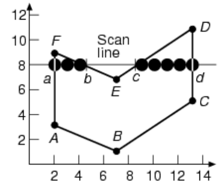
Algorytm z punktem środkowym

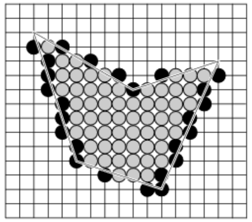
```
void MidpointCircle(int radius, int value)
{
  int x, y;
  float d;

  x = 0;                                 /* Initialization */
  y = radius;
  d = 5.0/4 - radius;
  CirclePoints(x, y, value);
  while(y > x){
       if(d < 0){                        /* Select E */
              d += x*2.0 + 3;
              x++;
       }else{                            /* Select SE */
              d += (x - y)*2.0 + 5;
              x++;
              y--;
       }
       CirclePoints(x, y, value);
  }
}
```

Wypełnianie wielokątów
Algorytm I

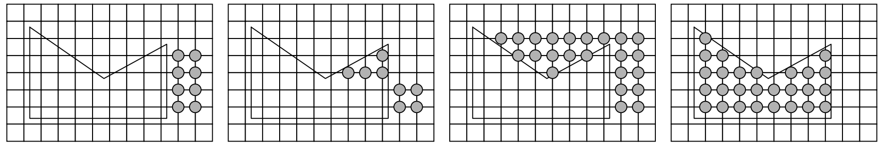
- Dla każdego poziomego wiersza śledzącego:
  1. Znajdź przecięcia wiersza śledzącego ze wszystkimi krawędziami wielokąta; pomiń krawędzie poziome
  2. Posortuj przecięcia według rosnącej wartości współrzędnej *x*
  3. Wypełnij między parą przecięć wszystkie piksele, które leżą wewnątrz wielokąta





Wypełnianie wielokątów
Algorytm II

- Dla każdej krawędzi wielokąta, z pominięciem krawędzi poziomych:
  1. Dla każdego poziomego wiersza śledzącego znajdź punkt przecięcia tego wiersza z krawędzią wielokąta
  2. Dopełnij (NOT) wszystkie punkty na prawo od punktu przecięcia

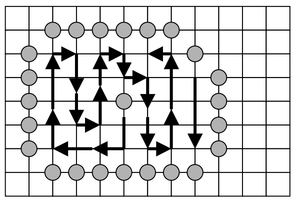


Wypełnianie wielokątów
Algorytm III

- Wielokąt musi być już wykreślony na ekranie
- Użytkownik wskazuje dowolny punkt wewnątrz wielokąta (x_0,y_0)

```
Push(x0,y0);
while stos niepusty {
  Pop(x,y);
  if (ReadPixel(x,y)<>value)
    WritePixel(x,y,value);

  if (ReadPixel(x+1,y)<>value)
    Push(x+1,y);
  if (ReadPixel(x,y+1)<>value)
    Push(x,y+1);
  if (ReadPixel(x-1,y)<>value)
    Push(x-1,y);
  if (ReadPixel(x,y-1)<>value)
    Push(x,y-1);
}
```

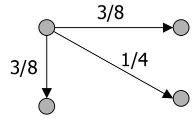


Aproksymacja półtonowa
Metoda progowa

```
for (y=0; y<ymax; y++)
  for (x=0; x<xmax; x++){
    if (ReadPixel(x,y)>threshold)
      WritePixel(x,y,maxvalue);
    else
      WritePixel(x,y,0);
  }
```

- modulacja progu sygnałem szumu (np. $A \cdot \sin\alpha x \cdot \sin\beta y$)

Aproksymacja półtonowa
Metoda Floyda-Steinberga

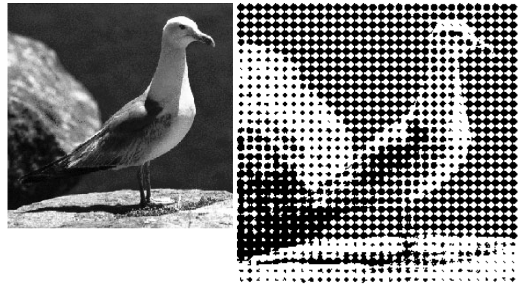


```
for (y=0; y<ymax; y++)
 for (x=0; x<xmax; x++){
  if (ReadPixel(x,y)>threshold) {
    error=ReadPixel(x,y)-maxvalue;
    WritePixel(x,y,maxvalue);
  }else{
    error=ReadPixel(x,y);
    WritePixel(x,y,0);
  }
  WritePixel(x+1,y,ReadPixel(x+1,y)+3/8*error);
  WritePixel(x,y-1,ReadPixel(x,y-1)+3/8*error);
  WritePixel(x+1,y-1,ReadPixel(x+1,y-1)+1/4*error);
 }
```

Aproksymacja półtonowa
Metoda komórkowa

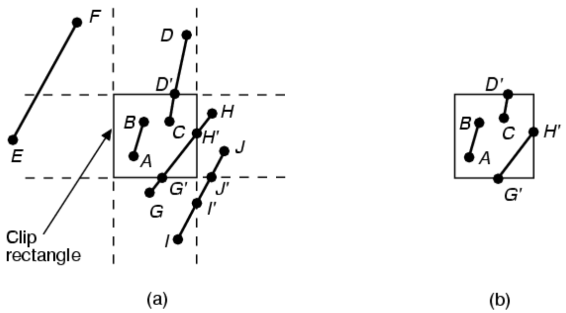
$$D_2 = \begin{bmatrix} 0 & 2 \\ 3 & 1 \end{bmatrix}$$

$$D_3 = \begin{bmatrix} 7 & 2 & 6 \\ 3 & 0 & 1 \\ 5 & 4 & 8 \end{bmatrix}$$



Element (i,j) komórki będzie zapalony, jeżeli ReadPixel(x,y)>D_n(i,j)

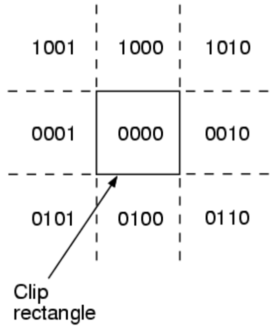
Obcinanie 2D - odcinki



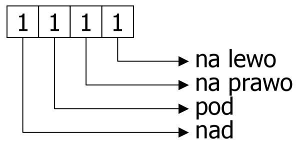
Algorytm Cohena-Sutherlanda

1. Odfiltruj proste przypadki
2. Wylicz punkty przecięcia pozostałych odcinków z krawędziami okienka

Filtrowanie prostych przypadków:

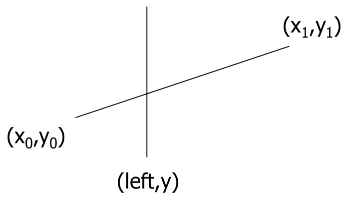


1. Zakoduj końce odcinka zgodnie z kodami obszarów
2. Jeżeli iloczyn logiczny (AND) tych kodów <>0, to odcinek może być pominięty (w całości poza oknem)
3. Jeżeli suma logiczna (OR)tych kodów = 0, to odcinek w całości mieści się w okienku



Algorytm Cohena-Sutherlanda

Wyliczanie punktów przecięcia odcinka z krawędziami okienka
(przykład dla lewej krawędzi)



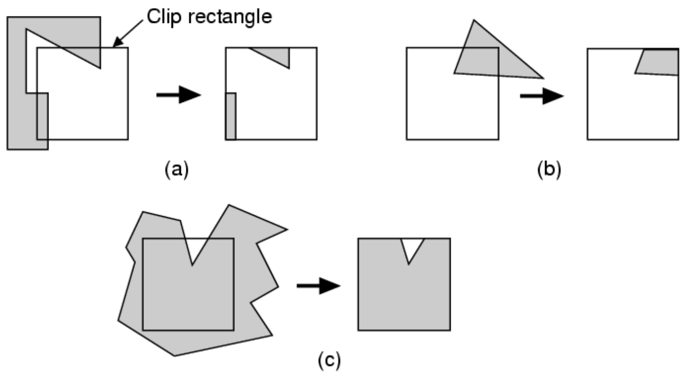
$$x = x_0 + s(x_1 - x_0)$$
$$y = y_0 + s(y_1 - y_0)$$

$$left = x_0 + s(x_1 - x_0)$$
$$s = \frac{left - x_0}{x_1 - x_0}$$
$$y = y_0 + s(y_1 - y_0)$$
$$y = y_0 + \frac{left - x_0}{x_1 - x_0}(y_1 - y_0)$$

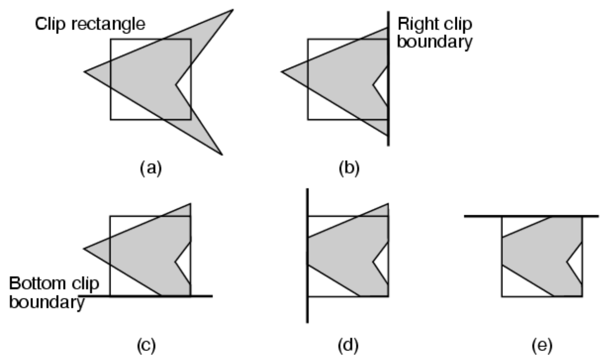
Algorytm Cohena-Sutherlanda

```
if (rcode & 8) { // bit 3 in region code
  x = left;
  y = y0+(y1-y0)*(left-x0)/(x1-x0);
} else if (rcode & 4) { // bit 2 in region code
  x = right;
  y = y0+(y1-y0)*(right-x0)/(x1-x0);
} else if (rcode & 2) { // bit 1 in region code
  x = x0 + (x1-x0)*(bottom-y0)/(y1-y0);
  y = bottom;
} else if (rcode & 1) { // bit 0 in region code
  x = x0 + (x1-x0)*(top-y0)/(y1-y0);
  y = top;
}
```

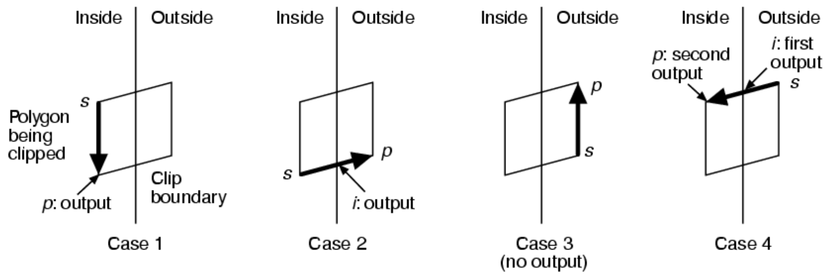
Obcinanie 2D - wielokąty



Algorytm Sutherlanda-Hodgmana



Algorytm Sutherlanda-Hodgmana



Algorytm Sutherlanda-Hodgmana

```
typedef struct vertex {
       float x, y;
} vertex;

typedef vertex edge[2];
typedef vertex vertexArray[MAX]; /* MAX is a declared constant */
```

```
void Intersect(vertex first, vertex second, vertex *clipBoundary,
                        vertex *intersectPt)
{
  if (clipBoundary[0].y == clipBoundary[1].y) { /*horizontal*/
       intersectPt->y = clipBoundary[0].y;
       intersectPt->x = first.x + (clipBoundary[0].y - first.y) *
              (second.x - first.x) / (second.y - first.y);
  } else { /*vertical*/
       intersectPt->x = clipBoundary[0].x;
       intersectPt->y = first.y + (clipBoundary[0].x - first.x) *
              (second.y - first.y) / (second.x - first.x);
  }
}
```

Algorytm Sutherlanda-Hodgmana

```
boolean Inside(vertex testVertex, vertex *clipBoundary)
{
  if (clipBoundary[1].x > clipBoundary[0].x) /*bottom*/
       if (testVertex.y >= clipBoundary[0].y)
              return TRUE;
  if (clipBoundary[1].x < clipBoundary[0].x) /*top*/
       if (testVertex.y <= clipBoundary[0].y)
              return TRUE;
  if (clipBoundary[1].y > clipBoundary[0].y) /*right*/
       if (testVertex.x <= clipBoundary[1].x)
              return TRUE;
  if (clipBoundary[1].y < clipBoundary[0].y) /*left*/
       if (testVertex.x >= clipBoundary[1].x)
              return TRUE;
  return FALSE;
}
```

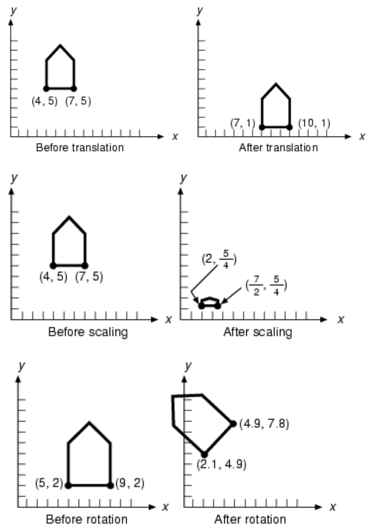
```
void Output(vertex newVertex, int *outLength, vertex *outVertexArray){
  (*outLength)++;
  outVertexArray[*outLength - 1].x = newVertex.x;
  outVertexArray[*outLength - 1].y = newVertex.y;
}
```

Algorytm Sutherlanda-Hodgmana

```
void SutherlandHodgmanPolygonClip(vertex *inVertexArray,
  vertex *outVertexArray, int inLength, int *outLength, vertex *clip_boundary)
{
  vertex s, p, i;
  int j;

  *outLength = 0;
  s = inVertexArray[inLength - 1];/* Start with the last vertex */
  for (j = 0; j < inLength; j++) {
        p = inVertexArray[j];
        if (Inside(p, clip_boundary)) {    /* Cases 1 and 4 */
                if (Inside(s, clip_boundary))
                        Output(p, outLength, outVertexArray); /* Case 1 */
                else { /* Case 4 */
                        Intersect(s, p, clip_boundary, &i);
                        Output(i, outLength, outVertexArray);
                        Output(p, outLength, outVertexArray);
                }
        } else if(Inside(s, clip_boundary)) { /* Cases 2 and 3 */
                Intersect(s, p, clip_boundary, &i); /* Case 2 */
                Output(i, outLength, outVertexArray);
        } /* No action for case 3 */
        s = p; /* Advance to next pair of vertices */
  }
}
```

Przekształcenia 2D



Przesunięcie (translacja)

$$x' = x + d_x$$
$$y' = y + d_y$$

Skalowanie

$$x' = s_x \cdot x$$
$$y' = s_y \cdot y$$

Obrót (rotacja)

$$x' = x \cdot \cos\theta - y \cdot \sin\theta$$
$$y' = x \cdot \sin\theta + y \cdot \cos\theta$$

Współrzędne jednorodne i macierze przekształceń 2D

Przesunięcie

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = T(d_x, d_y) \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Skalowanie

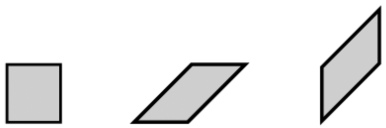
$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = S(s_x, s_y) \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Obrót

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = R(\theta) \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Dwa zestawy współrzędnych jednorodnych (x,y,W) i (x′,y′,W′) reprezentują ten sam punkt, gdy jeden jest wielokrotnością drugiego

Przekształcenie pochylające



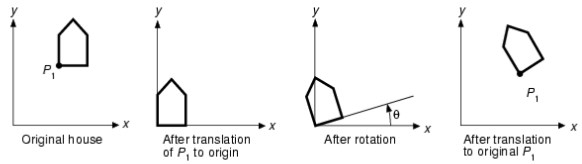
The unit cube sheared in the x direction

The unit cube sheared in the y direction

$$SH_x = \begin{bmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad SH_y = \begin{bmatrix} 1 & 0 & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

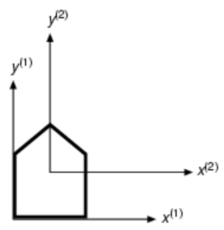
a, b – współczynniki proporcjonalności

Składanie przekształceń 2D
Przykład: obrót wokół dowolnego punktu



$$T(x_1, y_1) \cdot R(\theta) \cdot T(-x_1, -y_1) = \begin{bmatrix} 1 & 0 & x_1 \\ 0 & 1 & y_1 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot$$
$$\cdot \begin{bmatrix} 1 & 0 & -x_1 \\ 0 & 1 & -y_1 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & x_1(1-\cos\theta) + y_1 \sin\theta \\ \sin\theta & \cos\theta & y_1(1-\cos\theta) - x_1 \sin\theta \\ 0 & 0 & 1 \end{bmatrix}$$

Przekształcenia jako zmiana układu współrzędnych



$$P^{(2)} = M_{2\leftarrow 1} P^{(1)} = T(x_1, y_1) P^{(1)}$$