OpenGL feedback

OpenGL feedback

Mechanizm **sprzężenia zwrotnego** (ang. *feedback*) w OpenGL pozwala na przechwycenie przez aplikację efektów pracy biblioteki (współrzędne punktów, tekstur, kolory) bez wyświetlania ich na ekranie.

Informacje o parametrach przetworzonych obiektów zwracane są w formie tablicy wartości zmiennoprzecinkowych (bufor zwrotny).

OpenGL feedback – procedura postępowania

1) Określenie parametrów bufora dla danych zwrotnych z pomocą funkcji **glFeedbackBuffer()**.
2) Włączenie trybu sprzężenia zwrotnego z pomocą funkcji **glRenderMode(GL_FEEDBACK)**.
3) Rysowanie sceny (bez rasteryzacji).
4) Wygenerowanie pomocniczych znaczników w buforze zwrotnym z pomocą funkcji **glPassThrough()** - opcjonalnie.
5) Wyłączenie trybu sprzężenia zwrotnego przez wywołanie funkcji **glRenderMode(GL_RENDER)**. Funkcja zwraca liczbę danych zapisanych w buforze.
6) Przetwarzanie danych zwrotnych.

glFeedbackBuffer()

```
void glFeedbackBuffer(GLsizei size, GLenum type,
                                          GLfloat *buffer);
```

- `size` – maksymalna liczba wartości którą można umieścić w buforze (wielkość bufora);
- `type` – typ danych które będą zapisywane w buforze;
- `buffer` – wskaźnik na istniejący bufor RAM (tablica wartości float).

type	Współrz.	Kolor	Tekstury	RAZEM
GL_2D	X Y	-	-	2
GL_3D	X Y Z	-	-	3
GL_3D_COLOR	X Y Z	k	-	3 + k
GL_3D_COLOR_TEXTURE	X Y Z	k	4	7 + k
GL_4D_COLOR_TEXTURE	X Y Z W	k	4	8 + k

Format bufora

Po wypełnieniu bufora, znajdą się w nim **współrzędne ekranowe** każdego wierzchołka, jego **kolor wynikowy** (po wyliczeniach oświetlenia), itd.

Dla operacji z pikselami (np. glDrawPixels()), bufor zawiera dane każdego piksela bitmapy jako **czworokąt**.

Format bufora

Każdy blok danych w buforze zaczyna się od identyfikatora (poniżej), następnie danych wierzchołka(ów) w kolejności i liczbie ustalonej przez wywołanie **glFeedbackBuffer()**.

PRYMITYW	IDENTYFIKATOR	DANE
punkt	GL_POINT_TOKEN	wierzchołek
odcinek	GL_LINE_TOKEN lub GL_LINE_RESET_TOKEN	wierzchołek x2
wielokąt	GL_POLYGON_TOKEN	liczba, wierzchołek, wierzchołek, ...
bitmapa	GL_BITMAP_TOKEN	wierzchołek
prostokąt piksela	GL_DRAW_PIXEL_TOKEN lub GL_COPY_PIXEL_TOKEN	wierzchołek
marker	GL_PASS_THROUGH_TOKEN	liczba zmiennoprzecinkowa

glPassThrough()

```
void glPassThrough(GLfloat token);
```

Funkcja powoduje umieszczenie w buforze znacznika **GL_PASS_THROUGH_TOKEN** z wartością podaną jako „token”. Znaczniki takie mogą służyć np. do rozpoznawania danych kolejnych prymitywów w buforze.

Funkcja użyta w normalnym trybie renderingu nie daje żadnych efektów.

OpenGL feedback – przykład (1/3)

```
// Zmiana rozmiaru:
void rozm(GLsizei w, GLsizei h) {

   glViewport(0, 0, w, h);

   // Rzutowanie równoległe:
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-10, 10, -10, 10, 1, -1);
}
```

OpenGL feedback – przykład (2/3)

```
void rysuj() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);

    // Bufor zwrotny:
    float buff[128];
    glFeedbackBuffer(128, GL_3D, buff);
    // Tryb feedback:
    glRenderMode(GL_FEEDBACK);
    // "Rysowanie" punktów:
    glBegin(GL_POINTS);
        glVertex3f( 1,  1,  1);
        glVertex3f( 0,  0,  0);
        glVertex3f(-5,  5,  0);
        glVertex3f( 6,  0, -6);
    glEnd();
    // Przełączenie w tryb renderowania:
    int ile = glRenderMode(GL_RENDER);
    // Wyświetlenie wyników:
    wyswietlBufor(buff, ile);
    glFlush();
}
```

OpenGL feedback – przykład (3/3)

Efekty działania programu (rozmiar okna 400 x 400 pikseli):

```
ILE = 12
GL_POINT_TOKEN
X = 220
Y = 220
Z = 1
GL_POINT_TOKEN
X = 200
Y = 200
Z = 0.5
GL_POINT_TOKEN
X = 100
Y = 300
Z = 0.5
```

Ewaluator ze sprzężeniem zwrotnym

Ewaluator w OpenGL – typowe zastosowanie

```
// Współrzędne punktów kontrolnych:
GLfloat pkt[4*3] = {
    -4.0, -4.0, 0.0,     -2.0,  4.0, 0.0,
     2.0, -4.0, 0.0,      4.0,  4.0, 0.0};

void init() {
    // Określenie kształtu krzywej:
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, pkt);
    glEnable(GL_MAP1_VERTEX_3);
}

void draw() {
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_LINE_STRIP);
        for(int i = 0 ; i <= 30 ; i++)
            glEvalCoord1f((GLfloat) i/30.0);
    glEnd();
    glFlush();
}
```

Ewaluator sterujący animacją (1/7)

```
GLfloat k1[4*3] = {
   -8.0, -6.0, 0.0,     -7.0, -7.0, 0.0,  // Pierwsza krzywa
   -8.0,  6.0, 0.0,     -2.0,  6.0, 0.0
};
GLfloat k2[4*3] = {
   -2.0,  6.0, 0.0,      4.0,  6.0, 0.0,  // Druga krzywa
    9.0, -9.0, 0.0,      6.0,  4.0, 0.0
};

GLfloat animt = 0.0;        // Zmienna 0..1..0..1..
bool bAnimUp = true;        // Kierunek animacji
#define ANIM_SPEED   0.005f

// Zapamiętane aktualne rozmiary okna (viewport):
int win_szer, win_wys;
// Aktualne rozmiary bryły obcinania (glOrtho):
int box_szer, box_wys;
```

Ewaluator sterujący animacją (2/7)

```
void init() {
    // Uruchomienie ewaluatora:
    glEnable(GL_MAP1_VERTEX_3);
    // Kolor tła:
    glClearColor(0.85, 0.85, 0.85, 1.0);
}
```

Ewaluator sterujący animacją (3/7)

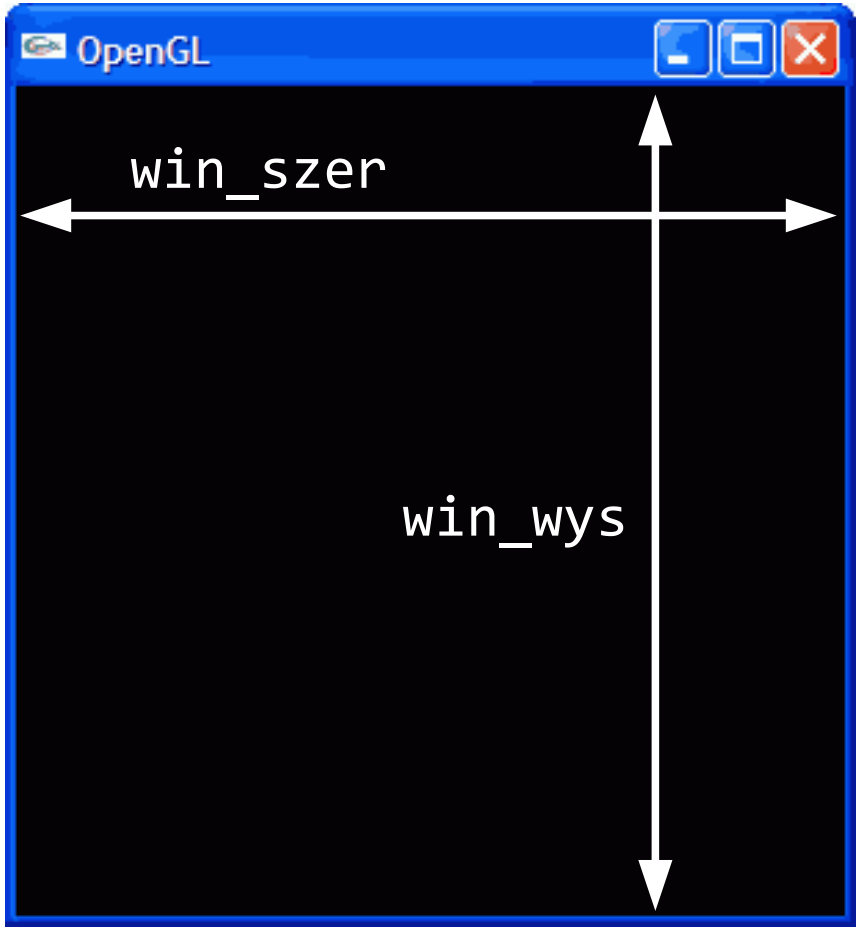
```
void ChangeSize(GLsizei w, GLsizei h) {
    if(h == 0)   h = 1;
    if(w == 0)   w = 1;
    // Zapamiętanie rozmiarów:
    win_szer = w;
    win_wys = h;
    // Ustawienia:
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    // Wyznaczenie i zapamiętanie rozmiarów bryły obcinającej:
    if(w < h) {
        box_szer = O_BOX;
        box_wys = O_BOX*h/w;
    }
    else {
        box_szer = O_BOX*w/h;
        box_wys = O_BOX;
    }
    // Ustawienie rzutowania równoległego:
    glOrtho(-box_szer, box_szer, -box_wys, box_wys, O_BOX, -O_BOX);
}
```

Ewaluator sterujący animacją (4/7)

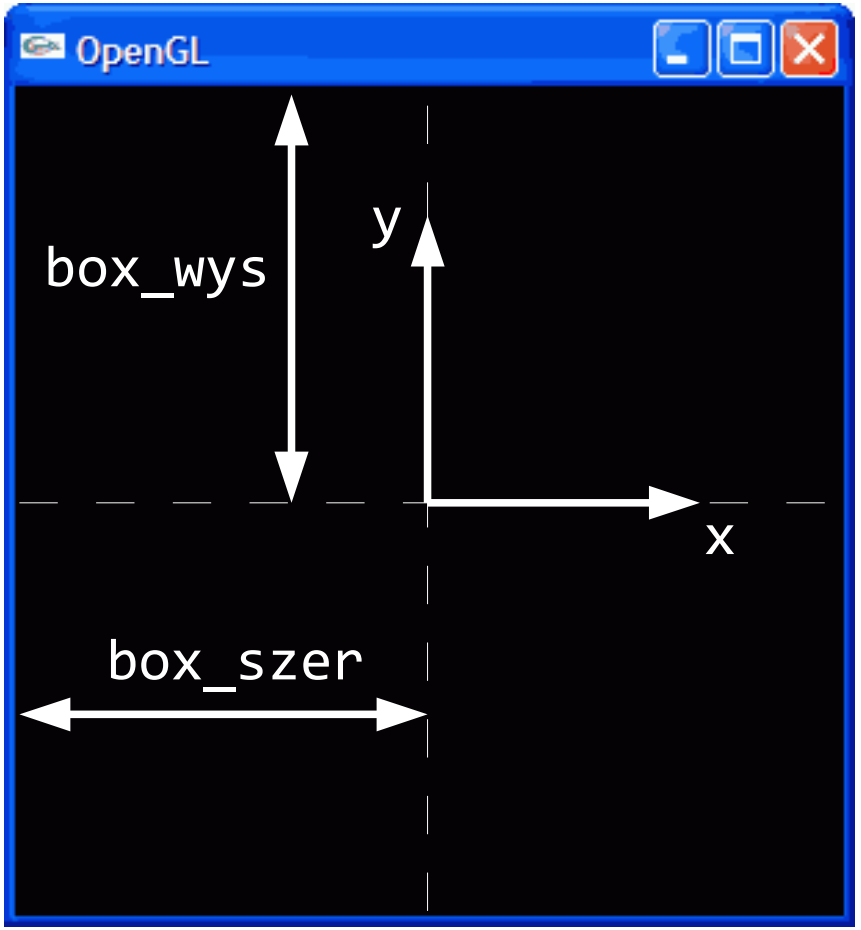
```
bool getBezierPos(float kr[], float t, float *x, float *y) {
    GLfloat buff[16];
    glFeedbackBuffer(16, GL_2D, buff);
    // Określenie kształtu krzywej:
    glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, kr);
    // Przechwycenie punktów, czyli przełączenie do trybu FEEDBACK:
    glRenderMode(GL_FEEDBACK);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glBegin(GL_POINTS);
        glEvalCoord1f(t);
    glEnd();
    // Przywrócenie normalnego rysowania:
    int ile = glRenderMode(GL_RENDER);

    if(ile>=3 && buff[0]==GL_POINT_TOKEN) {
        *x = 2*box_szer * buff[1] / win_szer - box_szer;
        *y = 2*box_wys * buff[2] / win_wys - box_wys;
        return(true);
    }
    return(false);
}
```

Konwersja współrzędnych ekranowych do 3D

2D



3D




```
X = 2*box_szer * Xe / win_szer - box_szer;
Y = 2*box_wys * Ye / win_wys - box_wys;
```

Xe, **Ye** – współrzędne ekranowe.

Ewaluator sterujący animacją (5/7)

```
void drawBezier(float kr[]) {
   // Określenie kształtu krzywej:
   glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4, kr);

   glLineWidth(3);
   glBegin(GL_LINE_STRIP);
   glColor3f(0.0, 0.0, 0.0);
      for(int i = 0 ; i <= 30 ; i++)
         glEvalCoord1f((GLfloat) i/30.0);
   glEnd();

   // Narysowanie punktów ze "strzałkami":
   drawPoints(kr, 4);
   // Kreski pomocnicze:
   glLineWidth(1);
   glBegin(GL_LINES);
      for(int i=0;i < 4;i++)
         glVertex2f(kr[i*3+0], kr[i*3+1]);
   glEnd();
}
```

Ewaluator sterujący animacją (6/7)

```
void RenderScene() {
    glClear(GL_COLOR_BUFFER_BIT);
    // Wyznaczenie współrzędnych X-Y w przestrzeni 3D:
    float x=0.0, y=0.0;
    bool res = false;
    // Dwie krzywe:
    if(animt <= 0.5)    res = getBezierPos(k1, animt*2, &x, &y);
    else                res = getBezierPos(k2, animt*2-1.0, &x, &y);
    if(res) {
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        // Narysowanie dużej kuli:
        glTranslatef(x, y, 0);
        glColor3f(1.0, 0.0, 0.0);    glutSolidSphere(1.5, 32, 32);
        // Narysowanie małej w środku:
        glColor3f(1.0, 0.4, 0.4);    glutSolidSphere(0.5, 16, 16);
    }
    // Narysowanie krzywych:
    glLoadIdentity();
    drawBezier(k1); drawBezier(k2);
    glutSwapBuffers();
}
```

Ewaluator sterujący animacją (7/7)

```
void ZegarFun(int val) {
   // Animacja:
   if(bAnimUp) {
      animt += ANIM_SPEED;
      if(animt >= 1.0) {
         animt = 1.0;
         bAnimUp = false;
      }
   }
   else {
      animt -= ANIM_SPEED;
      if(animt <= 0.0) {
         animt = 0.0;
         bAnimUp = true;
      }
   }
   glutPostRedisplay();
   glutTimerFunc(1000/ANIM_FPS, ZegarFun, 0);
}
```