FUNCŢII OPENGL PENTRU REDARE ANTI-ALIASING

Biblioteca OpenGL pune la dispoziţie două modalităţi de redare anti-aliasing: prin combinarea culorilor (*blending*) şi prin buffer de acumulare (*A-buffer*)

ANTI-ALIASING PRIN COMBINAREA CULORILOR

Această metodă dă rezultate de netezire numai dacă se renunţă la mecanismul Z-buffer (buffer-ul de adâncime) şi se desenează suprafeţele în ordinea în care sunt transmise bibliotecii. În această situaţie, culoarea unui pixel acoperit parţial de un poligon se obţine prin combinarea culorii poligonului cu culoarea pixelului, aflată în buffer-ul de culoare. În modul RGBA, OpenGL multiplică componenta A (alpha) a culorii poligoului cu ponderea de acoperire (raportul dintre aria acoperită de fragmentul pligonului şi aria pixelului). Această valoare poate fi folosită pentru ponderarea culorii în fiecare pixel prin combinare cu factorul `GL_SRC_ALPHA` pentru sursă şi factorul `GL_ONE_MINUS_SRC_ALPHA` pentru destinaţie. Pentru efectuarea acestor calcule mai este necesară validarea anti-aliasing-ului prin apelul funcţiei `glEnable()` cu unul din argumentele `GL_POINT_SMOOTH`, `GL_LINE_SMOOTH`, `GL_POLYGON_SMOOTH` pentru puncte, linii şi, respectiv, poligoane.

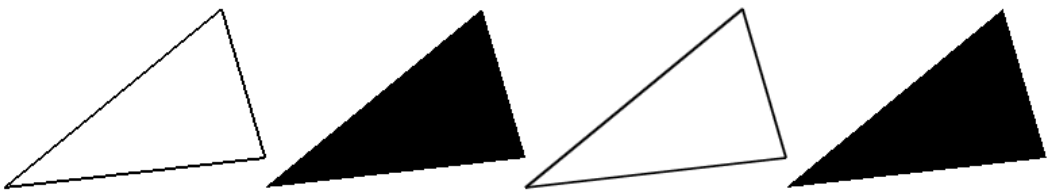
Exemplul 1.

Programul din acest exemplu este folosit pentru generarea imaginilor din figura 1 în care se desenează două triunghiuri cu şi fără anti-aliasing. Funcţiile `Init()` şi `Display()` ale programului dezvoltat sub GLUT sunt următoarele:

```
void Init()
{
	glClearColor(1.0, 1.0, 1.0, 1.0);
	glLineWidth(1.5);
	glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
}

void Display()
{
	int i;
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glColor3f(0.0, 0.0, 0.0);

	glPushMatrix();
		glTranslated(0.0, 0.0, -25.0);

		//triunghiuri cu aliasing
		glDisable(GL_BLEND);
		glDisable(GL_LINE_SMOOTH);
		glDisable(GL_POLYGON_SMOOTH);
		glPushMatrix();
			glTranslated(-8.0, 0.0, 0.0);
			glBegin(GL_LINE_LOOP);
				glVertex3d(-3, -3, 0);
				glVertex3d(3, -2, 0);
				glVertex3d(2, 3, 0);
			glEnd();
		glPopMatrix();
```

```
            glPushMatrix();
                  glTranslated(-2.0, 0.0, 0.0);
                  glBegin(GL_POLYGON);
                        glVertex3d(-3, -3, 0);
                        glVertex3d(3, -2, 0);
                        glVertex3d(2, 3, 0);
                  glEnd();
            glPopMatrix();

            //triunghiuri cu anti-aliasing
            glEnable(GL_BLEND);
            glEnable(GL_LINE_SMOOTH);
            glEnable(GL_POLYGON_SMOOTH);
            glPushMatrix();
                  glTranslated(4.0, 0.0, 0.0);
                  glBegin(GL_LINE_LOOP);
                        glVertex3d(-3, -3, 0);
                        glVertex3d(3, -2, 0);
                        glVertex3d(2, 3, 0);
                  glEnd();
            glPopMatrix();

            glPushMatrix();
                  glTranslated(10.0, 0.0, 0.0);
                  glBegin(GL_POLYGON);
                        glVertex3d(-3, -3, 0);
                        glVertex3d(3, -2, 0);
                        glVertex3d(2, 3, 0);
                  glEnd();
            glPopMatrix();

      glPopMatrix();
      glutSwapBuffers();

}
```

Deoarece anti-aliasing-ul se realizează prin combinarea culorilor, este necesară şi validarea combinării culorilor (`glEnable(GL_BLEND)`) pentru poligoanele cu anti-aliasing.



(a) (b)

Fig. 1. (a) Două poligoane cu aliasing. (b) Aceleaşi poligoane cu anti-aliasing

Acest mod de filtrare anti-aliasing este foarte simplist şi nu poate fi folosit decât pentru suprafeţe sau linii care nu se suprapun şi pot fi redate fără eliminarea suprafeţelor ascunse, aşa cum a fost situaţia în acest exemplu.

Dacă se redau obiecte tridimensionale complexe sau mai multe obiecte în scenă, este necesar să fie asigurată eliminarea suprafeţelor ascunse. Combinarea

culorilor nu funcţionează corect împreună cu mecanismul Z-buffer, iar generarea imaginilor fără eliminarea suprafeţelor ascunse este lipsită de realism.

Dacă se invalidează buffer-ul de adâncime, eliminarea suprafeţelor ascunse trebuie asigurată printr-un alt mecanism, de exemplu prin ordonarea suprafeţelor după adâncime. Dar eliminarea suprafeţelor ascunse prin ordonarea suprafeţelor după adâncime este un procedeu costisitor şi care nu poate fi implementat pentru scene complexe, cu un număr mare de obiecte. Soluţia acceptabilă ca performanţe şi rezultate pentru redarea anti-aliasing a scenelor complexe este folosirea unui buffer de acumulare (A-buffer).

ANTI-ALIASING PRIN EŞANTIONARE STOCASTICĂ

Principiul de bază al acestei metode este de a perturba în mod aleator poziţia punctelor de eşantionare. În acest fel, frecvenţele înalte ale imaginii, peste limita Nyquist, sunt transformate în zgomot, care apare în spectrul eşantioanelor în locul zgomotului de spectru transpus, şi poate fi eliminat prin filtrare.

În figura 2, o undă sinusoidală este eşantionată cu o frecvenţă mai mare decât limita Nyquist. Perturbaţia punctului de eşantionare introduce o eroare în amplitudinea eşantioanelor, care apare ca zgomot în spectrul imaginii eşantionate.

În cazul unei imagini, aceasta este eşantionată de mai multe ori în poziţii de eşantionare care sunt obţinute prin perturbare aleatoare faţă de centrul pixelului, pe o distanţă egală cu dimensiunea pixelului. Imaginile succesive obţinute pentru toate poziţiile de eşantionare sunt mediate pentru fiecare pixel (pe componente de culoare) rezultând imaginea finală, din care a fost eliminat zgomotul aleator datorat perturbării poziţiilor eşantionate. Fiecare imagine se creează într-un buffer de culoare şi apoi se adaugă ponderat la un buffer, care iniţial este şters. Acest buffer este numit *buffer de acumulare* sau eşantionare. În final, imaginea obţinută în buffer-ul de acumulare este transferată în buffer-ul de imagine.

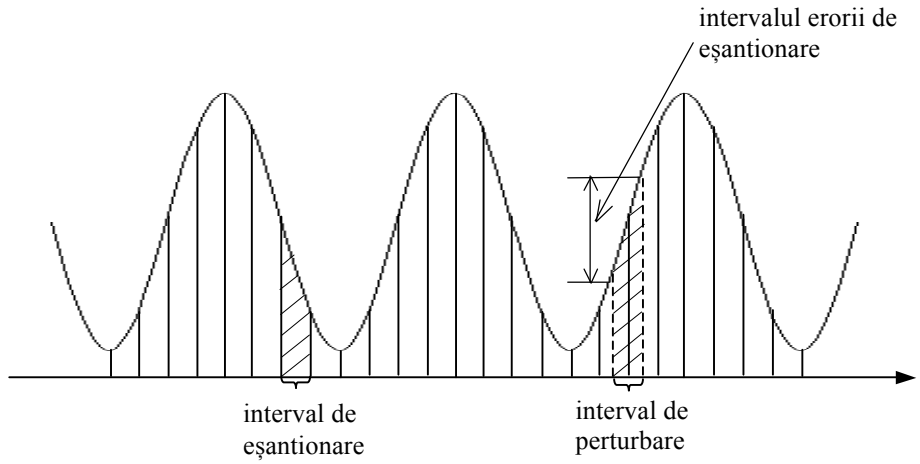


Fig. 2. Eşantionarea unei unde sinusoidale cu o frecvenţă peste limita Nyquist

Implementarea tehnicilor de anti-aliasing diferă foarte mult de la un sistem grafic la altul. O imagine de calitate (fără zgomot de spectru transpus - aliasing) necesită atât rezoluţii mari ale imaginii, care permit creşterea frecvenţei de eşantionare, cât şi filtrarea imaginii, prin care se limitează spectrul imaginii la o

valoare care asigură eşantionarea şi reconstrucţia fără zgomot (sau cu zgomot redus) a imaginii.

În sistemele grafice folosite în diferite aplicaţii de realitate virtuală, asigurarea unei filtrări anti-aliasing de calitate este o condiţie foarte importantă, mai ales în simulatoarele de antrenament. În cursul antrenamentelor de zbor, mişcarea dezordonată (*crowl*) a obiectelor, ca şi apariţia sau dispariţia obiectelor mici, datorată anti-aliasing-ului, au un efect negativ puternic asupra deprinderilor pe care trebuie să le însuşească piloţii. De aceea, s-au făcut eforturi deosebite pentru implementarea hardware a Z-buffer-ului, sau a texturării. Calitatea implementării procedeelor de anti-aliasing este un criteriu important de selecţie a unui sistem grafic, şi, bineînţeles, reprezintă un procent important din costul acestuia.

Biblioteca OpenGL implementează un buffer de acumulare care este folosit atât pentru redarea anti-aliasing a scenelor, cât şi pentru crearea altor efecte vizuale, ca iluminarea cu surse multiple de lumină sau estomparea imaginii unui obiect datorită mişcării acestuia (*motion blur*) .

Buffer-ul de acumulare se defineşte cu aceeaşi dimensiune ca şi buffer-ul de culoare şi buffer-ul de adâncime. În fiecare locaţie (x, y) a buffer-ului de acumulare se memorează culoarea unui pixel, corespunzător pixelului cu aceeaşi adresă (x, y) în buffer-ul de culoare. Asupra buffer-ului de acumulare se pot efectua operaţii prin apelul funcţiei:

```
void glAccum( GLenum op, GLfloat value);
```

Parametrul `op` selectează operaţia, iar parametrul `value` este un număr care este folosit în unele operaţii. Operaţiile posibile sunt `GL_ACCUM`, `GL_LOAD`, `GL_RETURN`, `GL_ADD` şi `GL_MULT`.

- `GL_ACCUM` citeşte fiecare pixel din buffer-ul curent de citire, selectat anterior cu funţia `glReadBuffer()`, multiplică valorile R,G,B,A ale pixelului cu valoarea `value` şi rezultatul îl adună la valoarea corespunzătoare (cu aceeaşi adresă de pixel) din buffer-ul de acumulare.
- `GL_LOAD` se execută asemănător cu operaţia `GL_ACCUM`, cu deosebirea că valorile rezultate înlocuiesc valorile existente în buffer-ul de acumulare.
- `GL_RETURN` preia valorile fiecărui pixel din buffer-ul de acumulare, le multiplică cu valoarea `value`, iar rezultatul îl înscrie în buffer-ul de scriere selectat anterior printr-o funcţie `glDrawBuffer()`.
- `GL_ADD` şi `GL_MULT` adună sau înmulţeşte valoarea componentelor R,G,B,A a fiecărui pixel din buffer-ul de acumulare cu valoarea `value`, rezultatul fiind depus înapoi în buffer-ul de acumulare. Pentru operaţia `GL_MULT`, `value` se limitează în intervalul `[-1.0, 1.0]`.

Înainte de începerea fiecărei operaţii de acumulare buffer-ul de acumulare se şterge prin apelul funcţiei `glClear(GL_ACCUM_BUFFER_BIT)`. Culoarea de ştergere a buffer-ului de acumulare trebuie să fie (0,0,0,0), şi se setează la iniţializare prin apelul funcţiei `glClearAccum(0.0, 0.0, 0.0, 0.0)`.

Buffer-ul de acumulare se foloseşte pentru implementarea anti-aliasing-ului prin eşantionare stocastică. Imaginea este eşantionată în poziţii de eşantionare care sunt perturbate aleator faţă de centrul pixelului, pe o distanţă egală cu dimensiunea pixelului. Imaginile succesive obţinute pentru toate poziţiile de eşantionare sunt

mediate pentru fiecare componentă de culoare a fiecărui pixel, rezultând imaginea finală, din care a fost eliminat zgomotul aleator datorat perturbării poziţiilor de eşantionare. Fiecare imagine se creează într-un buffer de culoare şi apoi se adaugă ponderat la buffer-ul de acumulare (iniţial şters) prin funcţia `glAccum(GL_ACCUM, 1/n)`, unde `n` este numărul de poziţii succesive de eşantionare. Imaginea rezultată în buffer-ul de eşantionare este transferată în buffer-ul de culoare prin funcţia `glAccum(GL_RETURN, 1.0)`. La generarea fiecărei imagini corespunzătoare unei poziţii de eşantionare se execută testul de adâncime (Z-buffer), deci imaginea finală elimină suprafeţele ascunse.

Aşa cum se vede în figura 3, efectele de aliasing sunt mult diminuate prin acest procedeu, dar, dacă este implementat soft, timpul de calcul este foarte mare, cel puţin de `n` ori mai mare decât timpul necesar pentru aceeaşi imagine redată fără anti-aliasing. Implementarea hardware a tehnicii de anti-aliasing este absolut necesară în grafica interactivă. Programul prin care s-a realizat această imagine este dat în exemplul următor.

Figura 3. Obiecte cu iluminare şi umbrire. La stânga, obiectul este desenat cu aliasing. La dreapta, obiectul este desenat cu anti-aliasng folosind un buffer de acumulare.

Exemplul 2.

Redarea anti-aliasing a obiectelor cu umbrire şi eliminarea suprafeţelor ascunse se poate implementa folosind buffer-ul de acumulare astfel:

```
#include <GL/freeglut.h>

#define ACCSIZE 16      //numarul de esantionari

static float W = 6;     //dimensiunea maxima a ferestrei
static float WX, WY;    //dimensiuni fereastra
static float N = -10;   //distanta de vizualizare near
static float F = 10;    //distanta de vizualizare far

void Init()
{
      GLfloat ambient[] = {0.4, 0.4, 0.4, 1.0};
      GLfloat diffuse[] = {1.0, 1.0, 1.0, 1.0};
      GLfloat specular[] = {1.0, 1.0, 1.0, 1.0};
      GLfloat position[] = {1.0, 0.2, 1.0, 0.0};
      GLfloat mat_ambient[] = {0.2, 0.2, 0.2, 1.0};
      GLfloat mat_diffuse[] = {0.9, 0.2, 0.1, 1.0};
```

```
	GLfloat mat_specular[] = {0.9, 0.6, 0.6, 1.0};
	GLfloat mat_shininess[] = {50.0};

	glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
	glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
	glLightfv(GL_LIGHT0, GL_SPECULAR, specular);
	glLightfv(GL_LIGHT0, GL_POSITION, position);

	glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
	glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
	glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
	glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);

	glEnable(GL_LIGHTING);
	glEnable(GL_LIGHT0);
	glEnable(GL_DEPTH_TEST);

	glClearAccum(0.0, 0.0, 0.0, 0.0);
	glClearColor(0.8, 0.8, 0.8, 1.0);
}

GLfloat jitter16[][2] = {
	{0.375, 0.4375}, {0.625, 0.0625},
	{0.875, 0.1875}, {0.125, 0.0625},
	{0.375, 0.6875}, {0.875, 0.4375},
	{0.625, 0.5625}, {0.375, 0.9375},
	{0.625, 0.3125}, {0.125, 0.5625},
	{0.125, 0.8125}, {0.375, 0.1875},
	{0.875, 0.6875}, {0.875, 0.0625},
	{0.125, 0.3125}, {0.625, 0.8125}
};

void Display()
{
	int i;
	glClear(GL_ACCUM_BUFFER_BIT);
	GLint viewport[4];
	glGetIntegerv(GL_VIEWPORT, viewport);

	//Redarea cu anti-aliasing a obiectului
	for(i = 0; i < ACCSIZE; i++)
	{
		glPushMatrix();
			glTranslatef(jitter16[i][0]*2*WX/viewport[2],
				jitter16[i][1]*2*WY/viewport[3], 0.0);
			glTranslated(1.5, -1, -8);
			glRotatef(15.0, 1.0, 0.0, 0.0);
			glRotatef(20.0, 0.0, 1.0, 0.0);
			glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);
			glutSolidTeapot(1.0);
		glPopMatrix();
		glAccum(GL_ACCUM, 1.0/ACCSIZE);
	}
	glAccum(GL_RETURN, 1.0);
	//Obiectul redat fara anti-aliasing
	glPushMatrix();
		glTranslated(-1.5, -1.0, -8.0);
		glRotatef(15.0, 1.0, 0.0, 0.0);
		glRotatef(20.0, 0.0, 1.0, 0.0);
		glutSolidTeapot(1.0);
	glPopMatrix();
```

```
	glFlush();

}

void Reshape(int w, int h)
{
	h = (h == 0) ? 1 : h;
	glViewport(0, 0, w, h);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	if(w <= h)
	{
		WX = W / 2;
		WY = W * (GLfloat)h / (GLfloat)(w * 2);
	}
	else
	{
		WX = W * (GLfloat)w / (GLfloat)(h * 2);
		WY = W / 2;
	}
	glOrtho(-WX, WX, -WY, WY, N, F);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}


void main(int argc, char* argv[])
{
	glutInit(&argc, argv);
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH |
			GLUT_ALPHA | GLUT_ACCUM);
	glutInitWindowSize(500, 500);
	glutInitWindowPosition(100, 100);
	glutCreateWindow("Acumulare");
	Init();
	glutDisplayFunc(Display);
	glutReshapeFunc(Reshape);
	glutMainLoop();
}
```

În funcţia de iniţializare `Init()` se defineşte sistemul de iluminare (surse de lumină şi materiale), se stabilesc culorile de ştergere ale buffer-ului de culoare şi de acumulare şi se validează buffer-ul de adâncime. Unele din comenzile din iniţializare nu au mai fost introduse dacă s-au folosit valorile implicite ale bibliotecii OpenGL. De exemplu, în execuţia cu un singur buffer de culoare (stabilită la iniţializare), buffer-ul de citire şi de scriere curent este implicit buffer-ul de culoare `GL_FRONT`, şi nu s-au mai apelat funcţiile `glReadBuffer()` şi `glDrawBuffer()`.

Perturbarea poziţiei de eşantionare se obţine prin perturbarea poziţiei centrului porţii de afişare. În acest exemplu s-a folosit o proiecţie ortografică în care transformările inverse de la poarta de afişare în sistemul universal se execută mai simplu. Fereastra este definită ca o fereastră simetrică, de dimensiuni `2*WX`, `2*WY`, calculate în funcţia `Reshape()`. Perturbaţia grilei de eşantionare se obţine prin modificarea centrului porţii cu valorile aleatoare $(\varepsilon_x, \varepsilon_y)$ care se citesc din tabelul

`jitter`. Acest tabel conţine perechi de valori în număr egal cu numărul de acumulări dorite (`ACCSIZE`), care sunt poziţii aleatoare pe suprafaţa unui pixel.

Perturbaţiile în poarta de afişare se transformă în perturbare a poziţiei centrului ferestrei de vizualizare prin scalare cu factorii de scalare `2*WX/viewport[2]`, `2*WY/viewport[3]`. Aceste deplasări ale poziţiei centrului ferestrei de vizualizare sunt echivalente cu deplasări în sens invers ale obiectelor scenei. În proiecţia ortografică, deplasarea obiectelor care conduce la deplasarea centrului porţii de afişare şi, deci, la perturbarea poziţiei grilei de eşantionare se obţine prin instrucţiunea:

```
glTranslatef(jitter16[i][0]*2*WX/viewport[2],
             jitter16[i][1]*2*WY/viewport[3], 0.0);
```

În proiecţia perspectivă, deplasarea obiectelor trebuie să fie calculată în funcţie de matricea de proiecţie perspectivă.