9. Transparenţa

9.1. Suport teoretic

Până acum s-a utilizat în aplicaţii cea de a patra componentă de culoare −alfa− dar s-a dat de fiecare dată valoarea 1 şi nu s-a discutat în mod special felul în care această valoare poate fi utilizată. Valorile alfa sunt furnizate cu funcţia `glColor#()` care specifică culoarea curentă, cu funcţia `glClearColor()` care specifică culoarea de ştergere şi atunci când se specifică anumiţi parametrii de iluminare cum ar fi intensitatea surselor de lumină cu funcţia `glLight#()` sau proprietăţi de material cu funcţia `glMaterial#()`. Nu s-a arătat însă până acum care este felul în care valoarea alfa afectează ceea ce se desenează. Dacă este activat blending-ul (amestecare) atunci culoarea fragmentului care se procesează va fi combinată cu valoarea pixelilor memoraţi deja în buffer-ul de cadru. Combinarea apare după ce scena care trebuie reprezentată a fost rasterizată şi convertită în fragmente, dar înainte ca pixelii finali să fie înscrişi în buffer-ul de cadru. Valorile alfa pot fi de asemenea utilizate în testul alfa pentru a accepta sau respinge un fragment în funcţie de valoarea sa alfa (**testul alfa**).

Dacă nu se face operaţia de amestecare (blending) atunci fiecare nou fragment se suprascrie peste valorile culorii existente deja în buffer-ul de cadru, ca şi cum fragmentul ar fi opac. Folosind combinarea (blending) se poate controla cât anume din culoarea existentă se va combina cu noua valoare a fragmentului. În acest fel valoarea alfa poate fi utilizată pentru a crea fragmente transparente, care lasă să se vadă ceva din ceea ce s-a memorat anterior pentru pixelul respectiv.

9.1.1. Factorii sursă şi destinaţie

Fragmentul şi pixelul au fiecare un factor care controlează contribuţia lor la culoarea finală a pixelului: factorul sursă, care este utilizat pentru a scala culoarea fragmentului care vine, şi factorul de amestecare destinaţie, care scalează pixelii citiţi din buffer-ul de cadru. Procesul de combinare a culorii fragmentului procesat (sursă) cu a culorii pixelului aflat deja în buffer-ul de cadru (destinaţie) se face în două etape (figura 1). Mai întâi trebuie specificat cum se calculează factorii sursă şi destinaţie. Aceşti factori au patru componente (pentru R,G,B,A) care sunt multiplicaţi cu fiecare componentă sursă şi destinaţie. Rezultatele multiplicărilor sunt adunate. Matematic această operaţie poate fi exprimată astfel:

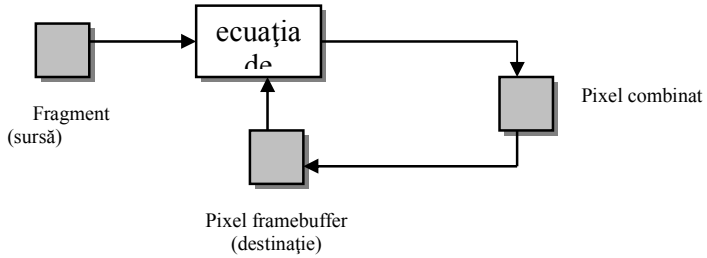

Figura 1

```
(SrRs+DrRd, SgGs+DgGd, SbBs+DbBd, SaAs+DaAd)                    (
```

unde: cvadruplul `(Sr, Sg, Sb, Sa)` reprezintă factorii pentru sursă
cvadruplul `(Dr, Dg, Db, Da)` reprezintă factorii pentru destinaţie
cvadruplul `(Rs, Gs, Bs, As)` reprezintă culoarea fragmentului sursă
cvadruplul `(Rd, Gd, Bd, Ad)` reprezintă culoarea fragmentului destinaţie

Să vedem acum care sunt funcţiile care sunt utilizate pentru amestecarea culorilor. În primul rând trebuie activată amestecarea folosind:

```
glEnable(GL_BLEND);
```

Dezactivarea se face cu `glDisable(GL_BLEND)`.
Factorii pentru sursă şi pentru destinaţie sunt furnizaţi folosind funcţia `glBlendFunc()`:

```
void glBlendFunc(GLenum sfactor, GLenum dfactor);
```

Argumentul `sfactor` arată cum se calculează factorul de amestecare al sursei; argumentul `dfactor` arată cum se calculează factorul de amestecare al destinaţiei. Se impune ca factorii de amestecare să fie cuprinşi în intervalul [0, 1]. În acest fel după combinare culoarea fragmentului se încadrează în domeniul [0, 1]. În tabelul 1 se dau valorile care pot fi luate de argumentele `sfactor` şi `dfactor`.

Tabelul 1 - Factorii de amestecare sursă şi destinaţie

Parametrul	**Relevanţa**	**Calcularea factorului de amestec**
`GL_ZERO`	sursă sau destinaţie	(0, 0, 0, 0)
`GL_ONE`	sursă sau destinaţie	(1, 1, 1, 1)
`GL_DST_COLOR`	sursă	(R_d, G_d, B_d, A_d)
`GL_SRC_COLOR`	destinaţie	(R_s, G_s, B_s, A_s)
`GL_ONE_MINUS_DST_ COLOR`	sursă	(1, 1, 1, 1)- (R_d, G_d, B_d, A_d)
`GL_ONE_MINUS_SRC_COLOR`	destinaţie	(1, 1, 1, 1)- (R_s, G_s, B_s, A_s)
`GL_SRC_ALPHA`	sursă sau destinaţie	(A_s, A_s, A_s, A_s)
`GL_ONE_MINUS_SRC_ALPHA`	sursă sau destinaţie	(1, 1, 1, 1)- (A_s, A_s, A_s, A_s)
`GL_DST_ALPHA`	sursă sau destinaţie	(A_d, A_d, A_d, A_d)
`GL_ONE_MINUS_DST_ALPHA`	sursă sau destinaţie	(1, 1, 1, 1)- (A_d, A_d, A_d, A_d)
`GL_SRC_ALPHA_SATURATE`	sursă	(f, f, f, f); f=min(A_s, 1-A_d)

9.1.2. Exemple de utilizare a amestecului

1. O modalitate de a desena o imagine care compune jumătate dintr-o imagine şi jumătate din altă imagine, egal amestecate, este de a seta factorul sursă ca `GL_ONE` şi de a desena prima imagine, apoi a seta factorii sursă şi destinaţie ca `GL_SRC_ALPHA` şi a desena cea de a doua imagine cu valoarea alfa de 0.5. Dacă imaginea finală trebuie să amestece în proporţie de 0.75 prima imagine şi 0.25 cea de a doua imagine, se va desena prima imagine ca mai înainte şi cea de a doua cu valoarea alfa de 0.25 dar cu factorul sursă setat `GL_SRC_ALPHA` şi cel destinaţie `GL_ONE_MINUS_SRC_ALPHA`. Această pereche de factori reprezintă probabil cea mai obişnuită utilizare a operaţiei de amestecare.

2. Pentru a amesteca în mod egal trei imagini, factorul destinaţie se setează ca `GL_ONE` şi factorul sursă ca `GL_SRC_ALPHA`. Se desenează fiecare dintre imagini cu o valoare alfa egală cu 0.3333333. Cu această tehnică, fiecare imagine are doar o treime din intensitatea originală, ceea ce este vizibil când imaginile nu se suprapun.
3. Să presupunem că se creează o aplicaţie de tipul aplicaţiei "Paint" din Windows şi se doreşte obţinerea unei pensule care adaugă în mod gradat culoare astfel că la fiecare atingere a pensulei se amestecă puţin mai multă culoare cu ceea ce există deja în imagine (de pildă, 10 procente culoare cu 90 procente din imagine la fiecare trecere). Pentru a se realiza aceasta se va desena imaginea pensulei cu procentul alfa de 10 şi se va utiliza factorul sursă `GL_SRC_ALPHA` şi factorul destinaţie `GL_ONE_MINUS_SRC_ALPHA` (se poate crea o pensulă care adaugă mai multă culoare în centrul ei şi mai puţină culoare în margine, folosind valori diferite ale lui alfa). Pentru a implementa o pensulă care şterge (gumă de şters) se va seta culoarea pensulei la culoarea fundalului.
4. Funcţiile de amestecare care utilizează culorile sursă şi destinaţie − `GL_DST_COLOR` sau `GL_ONE_MINUS_DST_COLOR` pentru factorul sursă şi `GL_SRC_COLOR` sau `GL_ONE_MINUS_SRC_COLOR` pentru factorii destinaţie − permit în mod efectiv modularea fiecărei componente de culoare în mod individual. Această operaţie este echivalentă cu aplicarea unui filtru simplu − spre exemplu, multiplicarea componentei roşu cu 80 procente, a componentei verzi cu 40 procente, şi a componentei albastre cu 72 procente va simula vizualizarea scenei printr-un filtru fotografic care blochează 20% din lumina roşie, 60% din lumina verde şi 28% din lumina albastră.
5. Presupunem că doriţi o imagine compusă din suprafeţe transparente, care se ascund una pe cealaltă şi toate acoperind un fundal netransparent. Presupunem că suprafaţa cea mai din spate transmite 80% din lumina din spatele ei, următoarea transmite 40% şi cea mai apropiată transmite 90%. Pentru realizarea acestei imagini se desenează mai întâi fundalul cu factori impliciţi pentru sursă şi destinaţie iar apoi se modifică factorul sursă la `GL_SRC_ALPHA` şi cel destinaţie la `GL_ONE_MINUS_SRC_ALPHA`. Apoi se desenează suprafaţa transparentă cea mai îndepărtată cu valoarea alfa de 0.2, suprafaţa mijlocie cu valoarea 0.6 şi în final suprafaţa cea mai apropiată cu valoarea 0.1.

Aplicaţia 1:

În exemplul amestec.c se vor amesteca culorile roşu şi verde iar apoi verde şi roşu. Obiectele care vor primi aceste culori sunt nişte simple primitive OpenGL, dreptunghiuri. Imaginea finală va conţine patru pătrate diferit colorate. Pătratele din colţul stânga-jos şi dreapta-sus vor fi desenate de două ori, de fiecare dată cu alta culoare. Se vor folosi cele două culori specificate, roşu şi verde dar în ordine diferită. Valoarea pentru alfa va avea de fiecare dată aceeaşi valoare şi anume 0.75. Factorii de amestecare vor fi: pentru sursă `GL_SRC_ALPHA`, pentru destinaţie `GL_ONE_MINUS_SRC_ALPHA`.

Să se explice de ce diferă culorile pătratului stânga-jos de cel dreapta-sus. Să se calculeze conform formulei 1 care sunt culorile finale ale acestor pătrate? Folosind această aplicaţie să se verifice efectul celorlalte combinaţii posibile exemplificate în tabel.

9.1.3. Eliminarea suprafeţelor ascunse şi transparenţa

Aşa cum s-a văzut din exemplul anterior ordinea desenării obiectelor ale căror culori se amestecă are efect asupra culorii finale. Atunci când se desenează obiecte transparente, pot fi obţinute aspecte diferite în funcţie de desenarea obiectelor dinspre spate spre faţă sau dinspre faţă spre spate. Pentru determinarea ordinii corecte trebuie ţinut seama şi de buffer-ul de adâncime. Buffer-ul de adâncime (z-buffer-ul) este utilizat pentru eliminarea suprafeţelor ascunse. El memorează distanţa dintre punctul de vizualizare şi porţiunea ocupată de un anumit pixel; atunci când o altă culoare candidează pentru pixelul respectiv, va fi înscrisă doar dacă obiectul căreia îi aparţine este mai aproape de punctul de vizualizare, caz în care valoarea adâncimii va fi memorată în buffer-ul de adâncime. Dacă algoritmul buffer-ului de adâncime este activat, porţiunile ascunse ale suprafeţelor nu sunt desenate motiv pentru care nu sunt utilizate la amestecare.
În mod obişnuit, se doreşte ca să se redea atât obiecte opace cât şi obiecte transparente în aceeaşi scenă, şi se doreşte de asemenea ca algoritmul de ascundere a suprafeţelor să fie activat astfel încât obiectele aflate în spatele obiectelor opace să fie eliminate. Dacă un obiect opac ascunde fie un obiect transparent fie un obiect opac se doreşte ca obiectul aflat la distanţa cea mai mare să fie eliminat. Dacă însă un obiect transparent se află în faţa altor obiecte se doreşte ca să se utilizeze amestecarea culorilor pentru simularea transparenţei. În general, pentru situaţia statică, chiar fără activarea algoritmului de ascundere, se poate stabili ordinea corectă de desenare ca aspectul să fie cel dorit. Dacă însă punctul de vizualizare îşi modifică poziţia sau dacă obiectele se mişcă problemele se complică.
Soluţia este de a se activa algoritmul de ascundere z-buffer şi buffer-ul de adâncime să fie făcut read-only atunci când se desenează obiectele transparente. Mai întâi se desenează toate obiectele opace, cu algoritmul z-buffer activat şi cu buffer-ul de adâncime în stare normală (poate fi scris/citit). Apoi valorile din z-buffer vor fi păstrate prin setarea buffer-ului de adâncime ca read-only. Se desenează apoi obiectele transparente. Deoarece algoritmul z-buffer este activat în continuare, dacă obiectele transparente se află în spatele obiectelor opace nu vor fi desenate, adâncimea lor fiind mai mare decât cea înscrisă în z-buffer. Dacă însă obiectele transparente se află în faţa obiectelor opace, ele nu vor elimina obiectele opace deoarece scrierea în z-buffer este blocată. Culoarea lor va fi însă amestecată cu a obiectelor opace, simulându-se astfel transparenţa.
Pentru a controla starea buffer-ului de adâncime se va utiliza funcţia `glDepthMask()`; dacă se transmite `GL_FALSE` ca argument buffer-ul z devine read-only iar dacă se transmite `GL_TRUE` se revine în starea normală.

Aplicaţia 2:

Să se deseneze două obiecte din biblioteca GLAUX (un cub şi o sferă) aflate unul în faţa celuilalt. Mai exact, cubul se află în faţa sferei. Sfera este opacă şi se desenează mai întâi, cu algoritmul z-buffer activat. Apoi se desenează cubul transparent, dar cu inactivarea scrierii în z-buffer. Culoarea cubului este amestecată cu cea a sferei. La apăsarea butonului stâng al mouse-ului se vizualizează scena din spate. În aceste condiţii, obiectul aflat mai în spate este cubul iar sfera se află mai în faţă.