eman ta zabal zazu

Universidad del País Vasco

Euskal Herriko Unibertsitatea

TESTURADUN POLIGONOAK

Lorea Loinaz Sagarzazu

2023ko irailaren 25a

Gaien Aurkibidea

1 SARRERA

Lan honen helburua testura duten triangeluak marraztuko dituen aplikazioa sortzea da. Lehenengo atal honetan, ordea, aplikazioaren amaierako funtzionalitatera iristeko oinarrizkoak diren pausuak egin ditugu: triangeluak marraztu eta testurarekin bete.

Puntu bakoitzak, berari dagozkion (x, y, z) koordenatuak ditu, eta baita testurari dagozkion (u, v) koordenatuak ere. Hortaz, marraztu beharreko triangelu bakoitzeko, hiru (x, y, z, u, v) koordenatu izango ditugu. Koordenatu hauek testu fitxategi batetik irakurriko ditugu.

Testurari dagokionez, .ppm formatura bihurtutako irudi bat izango dugu eta aplikazioak hasieran kargatuko du.

1: Testura gisa erabilitako irudia.

2 TRIANGELUAK MARRAZTU

2.1 Triangeluen puntuak ordenatu

Triangeluak marrazten hasteko lehenengo pausua, jakinik triangeluko erpinak p1, p2 eta p3 direla, triangeluko erpin bakoitza beraien y koordenatuaren arabera ordenatzea izan da.

Horretarako, lehenik eta behin, punto motako hiru pointer berri definitu ditugu, bakoitzak, ordenean, triangeluko puntu bat seinalatzeko helburuarekin.

```
punto *pgoiptr, *pbeheptr, *perdiptr;
```

Pointer horiei dagozkien erpinak esleitzeko, nahikoak izan dira hiru baldintza. Lehenengoak p1.y eta p2.y konparatuko ditu eta bietatik handiena izango da pgoiptr-k seinalatuko[1] duena eta txikiena, ordea, pbeheptr-k, seinalatuko duena. Hurrengo pausua, p3.y pgoiptr->y balioarekin konparatzea izango da, p3 puntua gorago badago, goiko puntua erdikoa izatera pasatzeko eta p3 goikoa izatera pasatzeko. Hirugarren puntua ordea goikoa baino beherago badago, beheko puntuarekin konparatu beharko da eta dagokion lekuan kokatu.

```
if(p1.y>p2.y)
{
    pgoiptr = &p1;
    pbeheptr = &p2;
}else{
    pgoiptr = &p2;
    pbeheptr = &p1;
}
if(p3.y > pgoiptr->y)
{
    perdiptr = pgoiptr;
    pgoiptr = &p3;
}else{
    if(p3.y<pbeheptr->y)
    {
        perdiptr = pbeheptr;
        pbeheptr = &p3;
    }else
        perdiptr = &p3;
}
```

[1] Pointerrari erpinaren helbidea esleituko zaio, horrela pointerra erpinari dagokion memoriako helbidera begira egongo da

2.2 Koordenatu barizentrikoak eta interpolazioa

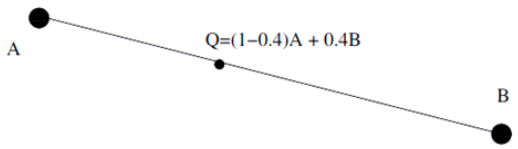
Puntuak ordenatu ondoren egin beharreko pausuak azaldu aurretik, lagungarriak izango diren bi kontzeptu azalduko ditut.

2.2.1 Koordenatu barizentrikoak

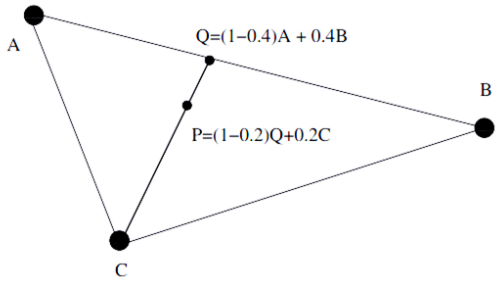
Koordenatu barizentrikoek, n-sinplex baten barnekaldea $[0, 1]$ tarteko $n+1$ zenbaki errealen konbinazio gisa adierazteko aukera eskaintzen dute non $\alpha_1+\alpha_2+...+\alpha_{n+1} = 1$ eta $0 \leq \alpha_1, \alpha_2, ..., \alpha_{n+1} \leq 1$ baldintzak betetzen diren.

Beste era batera esanda, koordenatu barizentrikoek, poligono edo zuzen baten barnealdeko puntu bat inguruan dituen erpinen distantzien portzentai gisa adierazteko aukera eskaintzen dute.

1-sinplex: lerro zuzena. $Q = \alpha_1 A + \alpha_2 B = (1-t)A + tB$



2-sinplex: hirukia. $P = \alpha_1 A + \alpha_2 B + \alpha_3 C$



2.2.2 Interpolazioa

Matematikan interpolazioa tarte batean magnitude baten gutxi gorabeherako balioa kalkulatzea da, tarte horretako balioetako batzuk ezagutzen direnean.

2.3 Ebaketa puntuak kalkulatu

Aurretik aipatutako bi kontzeptuak elkartuz, triangelua lerroz lerro mugatuko duten ebakidura puntuak oso modu errazean lortuko ditugu.

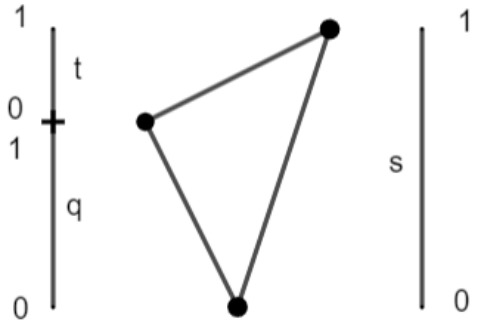
2.irudia erreferentzia gisa hartuta (ikus behean) zera da egin duguna: lehenik eta behin, hirukian hiru diferentzia kalkulatu ditugu y aldagaiaren arabera (t, q eta s). Horretaz gain, ebaketa puntuak kalkulatzeko triangelua bi zatitan banatu beharko dugu: hasieran t eta s erabiliko ditugu eta horren ondoren q eta s.

Koordenatu barizentrikoak erabiliz, badakigu t-ri dagokion zuzenaren $pebaketa = (1-t)pgoiptr + tperdptr$ izango dela, non t = [0,1] tarteko edozein puntu den. Aipatutakoa, beraz, puntuak interpolatzeko erabil dezakegu. Jakinik, triangelua lerroz lerro beteko dugula, t tarte hori tamaina berdinekoak izango diren n zatitan banatu nahi dugu. Tarte bakoitzak izango duen neurria kalkulatzeko zatiketa hau egingo dugu:

$$\frac{1}{pgoiptr->y + perdptr->y} = \Delta t$$

Ebaketa puntuak kalkulatzeko geratuko litzakeen pausu bakarra bi begizta sortzea litzake triangelua bi ataletan igarotzeko. Buelta bakoitzean t,q edota s-ri dagokien Δ balioa kenduko diegu, 0 izatera iritsi arte.

Hala ere, badago salbuespen nagusi bat: puntuak (bi edo gehiago) altura berdinean daudenean haien y balioaren arteko diferentzia 0 izango da, baina pantailan lerro bat agertzea nahi dugu. Hori konpontzeko, nahikoa izango da $\Delta > 1$ ezartzea, begizta behin egiteko aukera izango baitugu horrela.



2.irudia

Aipatutako eragiketak egiteko, lehenik eta behin Δ bakoitzari dagokien balioa esleitu diet eta, ondoren, bi begizten bidez, ebaketa puntuak interpolatu ditut.

```
if(pgoiptr->y!=perdiptr->y) delta12=1/(pgoiptr->y-perdiptr->y);
else delta12=2;

if(perdiptr->y!=pbeheptr->y)delta23=1/(perdiptr->y-pbeheptr->y);
else delta23=2;

s=1;
if(pgoiptr->y!=pbeheptr->y) delta13=1/(pgoiptr->y-pbeheptr->y);
else delta13=2;

for(t=1; t>=0; t-=delta12, s-=delta13)
{
    interpolatu_barizentroarekin(t, pgoiptr, perdiptr, &eb1);
    interpolatu_barizentroarekin(s, pgoiptr, pbeheptr, &eb2);

    if(eb1.x<eb2.x)
        dibujar_linea_z(eb1.y,eb1.x,eb1.z,eb1.u,eb1.v,eb2.x,eb2.z,eb2.u,eb2.v);
    else
       dibujar_linea_z(eb2.y,eb2.x,eb2.z,eb2.u,eb2.v,eb1.x,eb1.z,eb1.u,eb1.v);
}
for(q=1; q>=0; q-=delta23, s-=delta13)
{
    interpolatu_barizentroarekin(q, perdiptr, pbeheptr, &eb1);
    interpolatu_barizentroarekin(s, pgoiptr, pbeheptr, &eb2);

    if(eb1.x<eb2.x)
        dibujar_linea_z(eb1.y,eb1.x,eb1.z,eb1.u,eb1.v,eb2.x,eb2.z,eb2.u,eb2.v);
    else
        dibujar_linea_z(eb2.y,eb2.x,eb2.z,eb2.u,eb2.v,eb1.x,eb1.z,eb1.u,eb1.v);
}
```

```
void interpolatu_barizentroarekin(float t, punto *p1, punto *p2,
    punto *emaitzaptr)
{
    emaitzaptr->x= t*p1->x+(1-t)*p2->x;
    emaitzaptr->y= t*p1->y+(1-t)*p2->y;
    emaitzaptr->z= t*p1->z+(1-t)*p2->z;
    emaitzaptr->u= t*p1->u+(1-t)*p2->u;
    emaitzaptr->v= t*p1->v+(1-t)*p2->v;
}
```

Amaitzeko, dagoeneko hasita zegoen *dibujar_linea_z*() funtzioari dei egin diot beti ere dagokion ordenean (ezkerrerago zein puntu dagoen aztertu ostean).

2.4 Lerroa marraztu

Lerroa marrazteko funtzioak adierazitako bi puntuen artean dauden pixel guztiak marrazten ditu ezkerretik eskuinera. Hau egiteko ere, koordenatu barizentrikoak eta interpolazioa erabiliko ditugu eta aurrekoan bezalaxe Δt bat erabiliko dugu t aldagaia begizta baten barrua 1 baliotik 0 baliora pasa dadin, buelta bakoitzean t-=Δt eginez. Buelta bakoitzean gainera, u, v, x eta z koordenatu berriak kalkulatuko ditugu.

```
void  dibujar_linea_z(int linea,float c1x, float c1z, float c1u,
    float c1v,float c2x,float c2z,float c2u,float c2v)
{
    float xkoord,zkoord;
    float u,v;
    float t,deltat;
    unsigned char r,g,b;
    unsigned char *colorv;

    glBegin( GL_POINTS );
    deltat = 1.0/(c2x-c1x);

    for (t=1;t>=0;t-=deltat)
    {
        xkoord = t*c1x+(1-t)*c2x;
        zkoord = t*c1z+(1-t)*c2z;
        u =  t*c1u+(1-t)*c2u;
        v=  t*c1v+(1-t)*c2v;
        colorv=  color_textura(u, v);
        r= colorv[0];
        g=colorv[1];
        b=colorv[2];
        glColor3ub(r,g,b);
        glVertex3f(xkoord, linea, zkoord );

    }
    glEnd();
}
```

Baina horiek kalkulatu ostean, funtzio honek dagokion pixela kolore egokiarekin marraztu ahal izateko, testurari u eta v koordenatu horietan dagokion kolorea ondo hartzea ezinbestekoa da.

3 TRIANGELUAK MARGOTU

Aurretik egindako guztia egin ostean, testuraren kolorea egoki hartzea besterik ez da geratzen. Testuraren datu guztiak buffer batean gordeta daude (array batean, lerroz lerro baina datu guztiak jarraian) eta pixel bakoitzari hiru balio dagozkio, (r,g,b) balioak, hain zuzen ere.
Hori horrela izanik eta u eta v balioak parametro gisa izanik, bufferreko posizio bateko pointerra itzuliko dugu desplazamendua kalkulatuz.

Desplazamendu hori lortzeko, hainbat aldagai ezagutu behar dira:

- dimx eta dimy testurari dagozkion neurriak dira
- xind eta yind testuran bilatzen ari garen x eta y posizioak dira.
- u eta v aldagaiak guk lortu nahi dugun pixelaren koordenatuak dira beti ere haien balioa [0,1] tartean dagoelarik. Hala ere, testuraren eta pixelen v koordenatuak alderantzizkoak dira; hau da, bilatzen ari garen y balioa $y = 1 - v$ da. Horregatik yind kalkulatzeko orduan dimentsioaren modulua kalkulatu beharko dugu eta horren ostean, yind aldagaian $dimy - yind - 1$ balioa gorde beharko dugu.

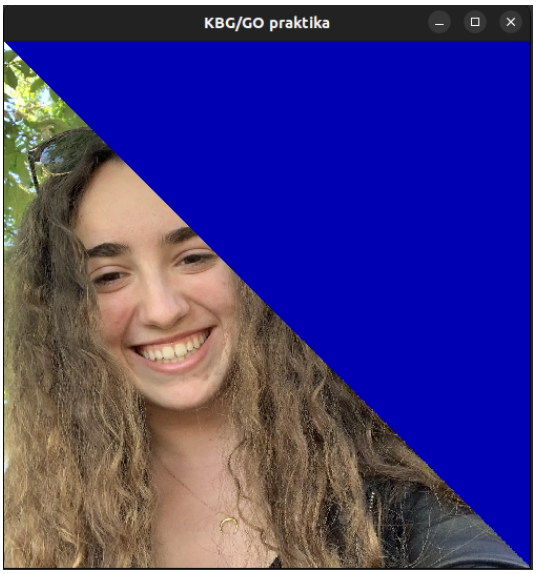
Hortaz, jakinik bufferrean guztia jarraian gordeta dagoela, desplazamendua 3*(dimx aldiz yind + xind) izango da.

```
unsigned char * color_textura(float u, float v)
{
    int desplazamendua;
    int xind, yind;
    char * lag;

    if(u<0) u=0;
    if(u>1) u=1;
    if(v<0) v=0;
    if(v>1) v=1;

    xind = u*(dimx-1);
    xind=xind % dimx;
    yind = v*(dimy-1);
    yind=yind % dimy;
    yind = dimy-yind-1;
    desplazamendua = yind*dimx + xind;

    lag = (unsigned char *)bufferra;
    return(lag+(3*desplazamendua));
}
```

4 ONDORIOAK

Aipatutako guztia laburbilduz, poligonoak betetzeko erabilitako algoritmoak, gehien batean, interpolazioa eta koordenatu barizentrikoak izan dira. Hauen bidez, era optimo batean, ebaketa puntuak eta lerro bakoitzeko pixel bakoitzaren koordenatuak aurkitu baitaitezke.

Algoritmo horiek aplikatzeko hiruko erregelak nahikoak izan dira eta, gainerako kalkuluak, era logikoan egiteko modukoak izan dira.



3.irudia: amaierako emaitza