Examen Teoria (2on parcial): Gràfics i Visualització de Dades
7 de juny de 2013

curs 2012-2013

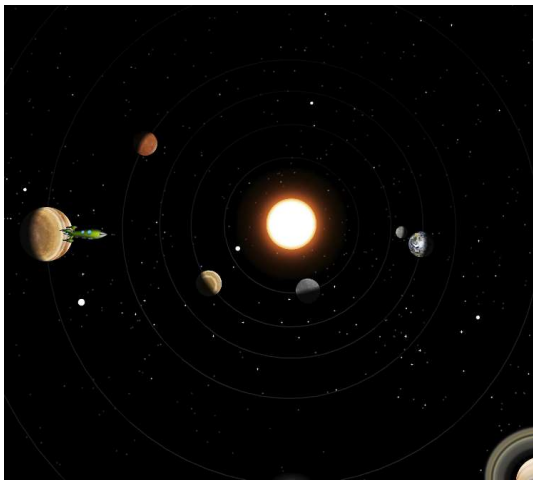
Problema (60 punts): Temps 2h

En aquest problema es parteix del planetari que heu implementat en les pràctiques del curs, on el planetari està format per planetes, un coet (situat inicialment en el punt 0,0,0 dret en l'eix Y+) i trajectòries circulars associades tant als planetes, com al coet. Cada planeta té una rotació i translació associada que es té en comte a cada *frame* quan s'activa el menú "Start Motion".

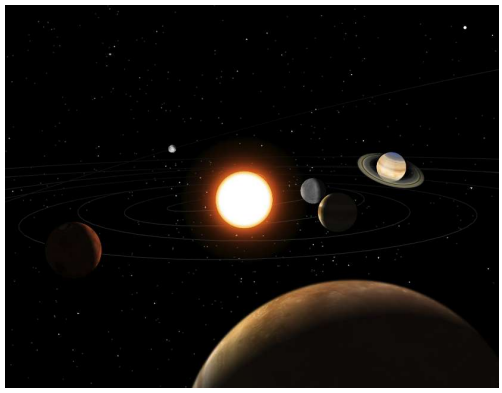
Es desitja dotar d'una nova utilitat a la pràctica suposant el següent funcionament. Inicialment, es carrega el planetari sense tenir activada la animació. En el *widget* principal es comença visualitzant tot el planetari centrat en el punt on està el Sol en una vista en planta (eixos X-Z), amb l'observador a una distància de 80 respecte el centre del planetari i en un *frame buffer* de 800x800 píxels. Aquesta vista l'anomenem Vista0. Suposa que el planetari sencer, incloses les trajectòries, estan dins del cub 100x40x100 i amb origen el punt (-50,-20,-50). En aquesta visualització, l'usuari selecciona un planeta, des del qual es vol simular el moviment del coet des del centre del planeta seleccionat cap al centre del Sol quan s'activa el menú de "Start Motion" (comença l'animació). Suposa que el centre del Sol està en el punt (0, 0, 0) en coordenades de món.

Per a tenir més control de la situació per part de l'usuari, quan comença l'animació:

1. Es re-visualitza en el mateix widget inicial de la Vista0, el planetari en planta (eixos X-Z) considerant que només es vol visualitzar la regió del planetari que comprèn el planeta seleccionat i la seva trajectòria al voltant del sol (Vista1)
2. S'obre un nou *widget* de 800x600 píxels que permet visualitzar el planetari simulant la visió que es té des del coet a cada *frame*, orientat cap al Sol (Vista2). En aquesta vista es visualitzarà una regió del planetari centrada en el Sol i de mida (4*r, 4*r, 4*r), on r és el radi del planeta seleccionat.



Vista 1 en seleccionar Saturn



Vista2 des del coet en el centre de Saturn

Tot suposant que tens una aplicació gràfica estructurada com el codi de la pràctica, respon les següents preguntes:

a) Defineix les classes i els atributs necessaris per a obtenir les vistes en les classes corresponents. (10 punts)

Cal modificar el disseny de classes de la pràctica per afegir un nou widget a mainwindow (el de la

Vista2). Així es tenen les classes:

- mainwindow: amb dos atributs widget, l'atribut Timer i l'atribut d'escena. L'escena es passarà per referencia al constructor de cada widget. En començar l'aplicació es crea el widget per la Vista0. En el mètode associat al callback del ratolí de la Vista0 es crea el widget de la Vista2, quan es selecciona el planeta.
- Dos widgets: un de les vistes Vista0 i Vista1 i un altre per la Vista2. Cadascun tindrà la seva càmera.
- Una escena: que serà compartida pels dos widgets. Serà el contenidor dels objectes de l'escena i cada objecte tindrà la seva trajectòria associada.

Quan arribi un event de Timer, es connectarà a un mètode de mainwindow que actualitzarà l'escena i farà un signal al widget de la Vista0/Vista1 i un signal al widget de la Vista2, en cas que estigui creada per tal d'actualitzar les visualitzacions. Aquest és una possible solució, encara que no la única.

b) Suposa que l'usuari selecciona el planeta Saturn que està centrat en el punt de món (35, 0, 0), té radi 5 i realitza una trajectòria circular centrada en el Sol, de radi 35. Defineix tots els atributs de les càmeres. Quins atributs cal modificar de la càmera de la Vista0 per calcular la Vista1? A partir dels atributs, defineix les transformacions window-viewport per a les 2 vistes que es calculen en el moment de començar l'animació (Vista1 i Vista2), suposant que utilitzes una projecció paral·lela. Cal aplicar alguna transformació geomètrica als objectes abans de la visualització inicial de les vistes Vista1 i Vista2? En cas necessari, des d'on s'hauria de cridar?. (20 punts)

Defineix tots els atributs de les càmeres.

Càmera de la Vista 0: Això no ho demana l'enunciat però es per centrar els conceptes.

VRP = (0,0,0)
view_vector = (0, 1, 0)
d = 80 (suficientment gran, per exemple 80)
Posició de l'observador: (0, 80, 0)
window: pmin = -50, -50, a = 100, h = 100
Els plans antero-posterior han de ser els que permetin veure tota l'escena: es poden posar a 0<zmin<=60 i zmax>=100
viewport: pmin = 0,0, a = 800, h = 800

Càmera de la Vista 1: En la Vista1, en relació a la Vista0, canvia la window.

VRP = (0,0,0)
view_vector = (0, 1, 0)
d = 80 (suficientment gran, per exemple 80)
posició de l'observador: (0, 80, 0)
window: es calcula la trajectoria màxima i mínima de Saturn tenint en comte el seu radi. Això dona una capsa centrada en el 0,0,0 i mirant només les coordenades x,z (ja que es en planta) dóna la següent window: pmin = -40, -40, a = 80, h = 80
Els plans antero-posterior han de ser els que permetin veure tota l'escena: es poden posar a zmin<=60 i zmax>=100, igual que la Vista 0.

viewport: pmin = 0,0, a = 800, h = 800

Càmera de la Vista 2:
posició de l'observador: centre de Saturn (35, 0, 0)

VRP = (0,0,0)
view_vector = (1, 0, 0)
d = 35
window: pmin = -10, -10, a = 20, h = 20
Els plans antero-posterior han de ser els que permetin veure tota l'escena: es poden posar a zmin<=10 i zmax>=90

viewport: pmin = 0,0, a = 800, h = 600

Quins atributs cal modificar de la càmera de la Vista0 per calcular la Vista1?

Només la window, segons les dades anteriors.

A partir dels atributs, defineix les transformacions window-viewport per a les 2 vistes que es calculen en el moment de començar l'animació (Vista1 i Vista2), suposant que utilitzes una projecció paral·lela.

- A partir dels atributs, defineix les transformacions window-viewport per a les 2 vistes que es calculen en el moment de començar l'animació (Vista1 i Vista2), suposant que utilitzes una projecció paral·lela: En la Vista1 i en la Vista2, es defineix la transformació window-viewport seguint la relació d'aspecte, general:

$$\frac{xw-(xini)}{amplada_w}=\frac{xv-0}{amplada_v}$$

$$\frac{yw-(yini)}{alcada_w}=\frac{alçada_v-(yv-0)}{alçada_v}$$

  També es pot utilitzar la fòrmula vista a teoria en el tema 4.

- A partir dels atributs, la transformació window-viewport de la Vista 1 seria:

```
glViewport(0, 0, 800, 800);
```

  En la vista en Alçada (XZ) la window té l'orígen a -40, -40 i té una amplada de (35*2 + 5*2) = 80 i una alçada de 80. Aixi, la conversió window-viewport a la vista de Perfil seria:

$$\frac{xw-(-40)}{80}=\frac{xv-0}{800} \qquad \frac{yw-(-40)}{80}=\frac{800-(yv-0)}{800}$$

  Això es pot posar en forma matricial si aïllem xv, yv en funció de xw, yw, quedant:

$$xv=\overset{Sx}{10}*xw+\overset{Tx}{400} \qquad yv=\overset{Sy}{-10}*yw+\overset{Ty}{0.5}$$

  Això suposa un escalat (Sx i Sy) i una translació (Tx, Ty) del punt xw, yw

$$(xv,yv,1)^T=\begin{bmatrix}1 & 0 & Tx\\ 0 & 1 & Ty\\ 0 & 0 & 1\end{bmatrix}*\begin{bmatrix}Sx & 0 & 0\\ 0 & Sy & 0\\ 0 & 0 & 1\end{bmatrix}*(xw,yw,1)^T$$

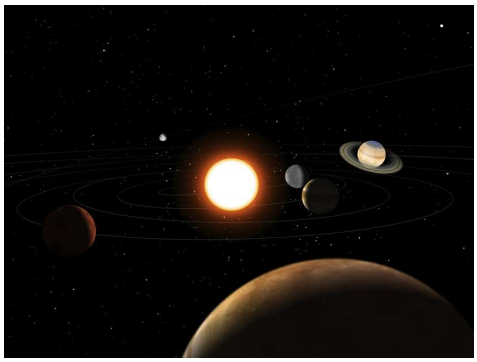
- A partir dels atributs definits anteriorment, la transformació window-viewport de la Vista 2 seria:
  glViewport(0, 0, 800, 600);

$$\frac{xw-(-10)}{20}=\frac{xv-0}{800} \qquad \frac{yw-(-10)}{20}=\frac{600-(yv-0)}{600}$$

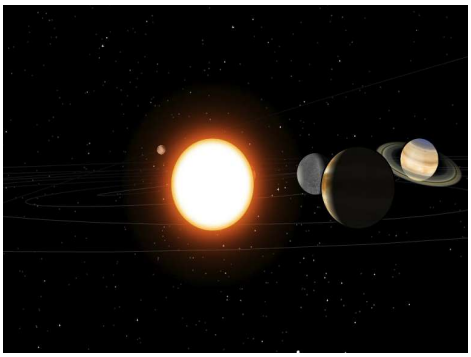
Cal aplicar alguna transformació geomètrica als objectes abans de la visualització inicial de les vistes Vista1 i Vista2? En cas necessari, des d'on s'hauria de cridar?.

Des del callback del ratolí que selecciona el planeta en el widget de la Vista 1, es crea la Vista 2. En aquest moment cal fer una translació del coet al punt 0,0,0, (però ja hi està situat i no caldria), aplicar una rotació al coet per situar-lo a l'eix X (rotació +90 en l'eix Z) i una translació per situar-lo a l'origen de Saturn (35,0,0). Els planetes, en aquest primer Frame, no s'han de moure de les seves posicions inicials.

c) A cada *frame*, com calcules les vistes Vista1 i Vista2? Indica els passos que s'han de realitzar a cada pas del *timer*, detallant els atributs que canvien, les transformacions necessàries i les transferències a la GPU que calen. Defineix les noves càmeres calculades a cada pas, en cas que sigui necessari actualitzar-les. Pots realitzar càlculs incrementals entre *frames*? Per què? Com es sincronitzen les vistes? En relació a la Vista2, suposa que el coet es mou sobre una trajectòria rectilínia que comença a l'origen del planeta seleccionat i acaba en el centre del sol. Aquesta trajectòria es constant en tota l'animació. (20 punts)



Frame 1 de la Vista2



Frame 2 de la Vista2

A cada crida al callback del timer s'ha de:

1. Actualitzar l'escena: això implica realitzar les TGS corresponents a rotació i translació de cada planeta i la translació del coet per la trajectòria rectilínia, un cert delta.
2. La càmera de la Vista1 no es re-calcula, en el widget associat a la Vista1 cal tornar a enviar a la GPU els planetes actualitzats i cridar a l'updateGL().
3. La càmera de la Vista2 es re-calcula. Aquí hi han dues opcions: o bé reduir la window i fer l'efecte de zoom en la projecció paral.lela o bé deixant la window constant i movent un delta la posició de l'observador i el VRP al llarg de la recta de la trajectòria del coet. En el primer cas, la window es pot calcular escalant la window anterior. En segon cas, cal canviar el plans de retallat antero-posterior per garantir la projecció correcta de l'escena. Com canvia aquesta càmera cal passar a la GPU la càmera de la Vista2. També cal passar els objectes transformats a la GPU, excepte el coet, ja que no es visualitza. Finalment cal fer l'updateGL() des del widget associat a la Vista2.

Les vistes es sincronitzen de forma automàtica quan s'actualitzen des del timer. Des del mètode de tractament del timer es fa un signal al Widget de la Vista1 i un signal al widget de la Vista 2.

En la Vista 1 es realitzen les següents accions:

actualitza_escena() : actualitza els moviments dels planetes i del coet. Actualitza l'escena compartida de les dues vistes
updateGL() : s'encarrega de tornar a passar els objectes a la GPU i dibuixar-los un a un

En el mètode del connect del signal de la Vista 2 es realitzarien les següents accions:

calculWindow(zoom): escala la window actual en un tant per cent calculat amb la distància del planeta des d'on està el coet al sol. Si la distancia del planeta del coet al sol és d, i el moviment del coet a cada frame és d', la window s'escala en un factor 1+d'/d.
calculCamera(centrada=true, retallat=true): recalcul de la camera amb la nova window i de les matrius associades. En aquest cas només calcdria recalcular la matriu projection
camera->toGPU() : envia la nova camera a la GPU
updateGL() : redibuixa l'escena, sense realitzar cap transformació als objectes de l'escena. Les transformacions de moviment dels planetes i del coet ja s'han realitzat des del widget de la Vista 1.

d) Suposa que es té una llum puntual situada en el Sol. Es desitja utilitzar el *toon shading* de color en la vista central i el *toon shading* de normals en la vista del coet, utilitzant 4 colors diferents per a cada planeta basats en el seu material difús K_d. Raona on es realitzen els càlculs i quins passos són necessaris. Quan cal transferir el material de cada planeta a la GPU? Implementa els *shaders* que es necessiten. Quina vista es veu més suau? Per què? (10 punts)

La diferència entre el toon-shading de color i toon-shading de normals és el lloc on es el calcula la intensitat. En el toon-shading de colors es calcula la intensitat enel vertex shader i es passa directament al fragment shader. El fragment shader només agafa el color interpolat i el mostra.

El vertex shader associat al toon-shading de colors de la vista dels planetes és el següent:

```
#if __VERSION__<130
#define IN varying
#define OUT varying
#else
#define IN in
#define OUT out
#endif

// vertex shader
IN  vec3 fN;

IN   vec4 vPosition;

OUT vec4 color;

uniform vec4 ambientLightEsc;

struct M
{
    vec4 ka;
    vec4 kd;
    vec4 ks;
```

```
    float coef_esp;
};

uniform M material;

const int numberOfLights = 1;

struct PerLight
{
    vec4 LightPosition;
    vec3 La;
    vec3 Ld;
    vec3 Ls;
    float lightAttenuationK;
    float lightAttenuationL;
    float lightAttenuationQ;
};

uniform PerLight light;

void main()
{
    vec3 lightDir;

    float intensity;

    vec3 pos = (ModelView * vPosition).xyz;

    lightDir = normalize(vec3(1.0, 1.0, 0.0));

    intensity = dot(lightDir,normalize(fN));

    if (intensity > 0.95)
           color = vec4(material.kd.r,material.kd.g,material.kd.b,1.0);
    else if (intensity > 0.5)
           color = vec4(material.kd.r*0.75,material.kd.g*0.75 ,material.kd.b*0.75,1.0);
    else if (intensity > 0.25)
           color = vec4(material.kd.r*0.5,material.kd.g*0.5 ,material.kd.b*0.5,1.0);
    else
           color = vec4(material.kd.r*0.25,material.kd.g*0.25 ,material.kd.b*0.25,1.0);

}
```

El fragment shader associat al toon-shading de colors seria:

```
in  vec4 color;
void main()
{
    gl_FragColor = color;
}
```

Pel toon-shading de normals en el vertex shader es faria la transformació dels punts amb la model-view i la matriu projection. No es calcularia cap color ni intensitat.

Aquest seria el fragment shader associat a toon-shading de normals de la vista del coet, on el calcul de la intensitat es fa en el fragment-shader.

```
// per-fragment interpolated values from the vertex shader
IN  vec3 fN;

OUT vec4 color;
```

```
uniform vec4 ambientLightEsc;

struct M
{
    vec4 ka;
    vec4 kd;
    vec4 ks;
    float coef_esp;
};

uniform M material;

const int numberOfLights = 1;

struct PerLight
{
    vec4 LightPosition;
    vec3 La;
    vec3 Ld;
    vec3 Ls;
    float lightAttenuationK;
    float lightAttenuationL;
    float lightAttenuationQ;
};

uniform PerLight light;

void main()
{
    vec3 lightDir;

    float intensity;

    lightDir = normalize(vec3(1.0, 1.0, 0.0));

    intensity = dot(lightDir,normalize(fN));

    if (intensity > 0.95)
           gl_FragColor = vec4(material.kd.r,material.kd.g,material.kd.b,1.0);
    else if (intensity > 0.5)
           gl_FragColor = vec4(material.kd.r*0.75,material.kd.g*0.75 ,material.kd.b*0.75,1.0);
    else if (intensity > 0.25)
           gl_FragColor = vec4(material.kd.r*0.5,material.kd.g*0.5 ,material.kd.b*0.5,1.0);
    else
           gl_FragColor = vec4(material.kd.r*0.25,material.kd.g*0.25 ,material.kd.b*0.25,1.0);

}
```

Quan cal transferir el material de cada planeta a la GPU?

Cal transferir els materials a la GPU per a cada planeta, per tal de recalcular el to pertinent a cada planeta.

Quina vista es veu més suau? Per què?

Es veuria una transició més suau en la vista del coet ja que s'interpolen les normals i no els colors. Es similar al que passa amb el calcul de Gouraud i el calcul de Phong-Blinn. La interpolació de normals dona un efecte més suau en les transicions de color, que la interpolació de color.