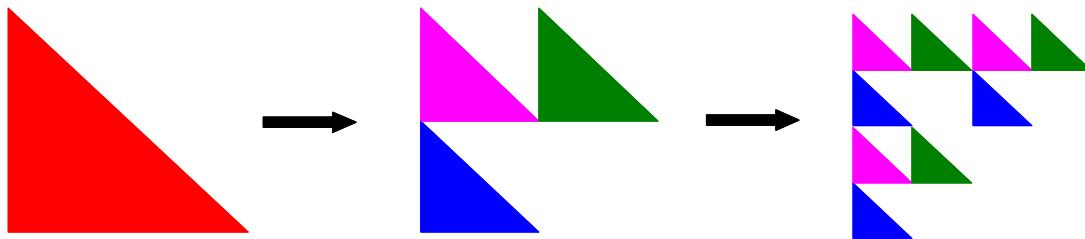


# Examen Informática Gráfica (14/09/16)

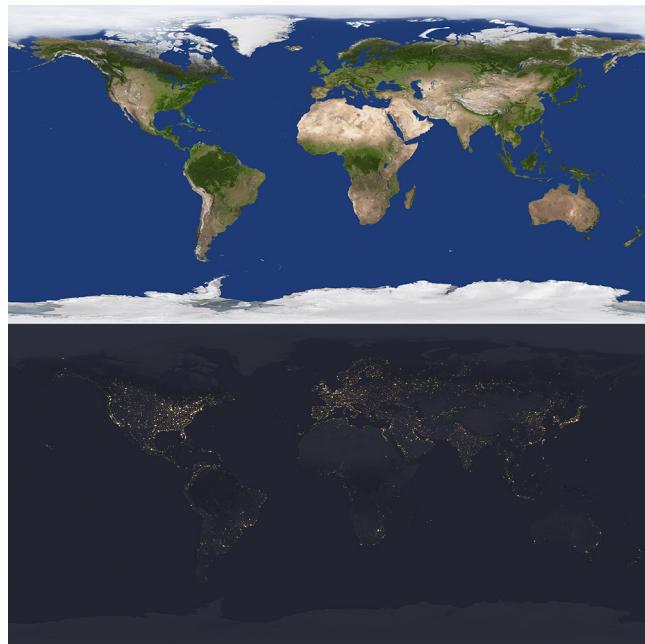
Nombre: \_\_\_\_\_ Grupo: \_\_\_\_\_

- 1) Programar usando C++ y OpenGL un procedimiento que de forma recursiva permita generar el siguiente patrón. Se indicará el número de niveles de la recursión mediante un parámetro. (2.5 pt)

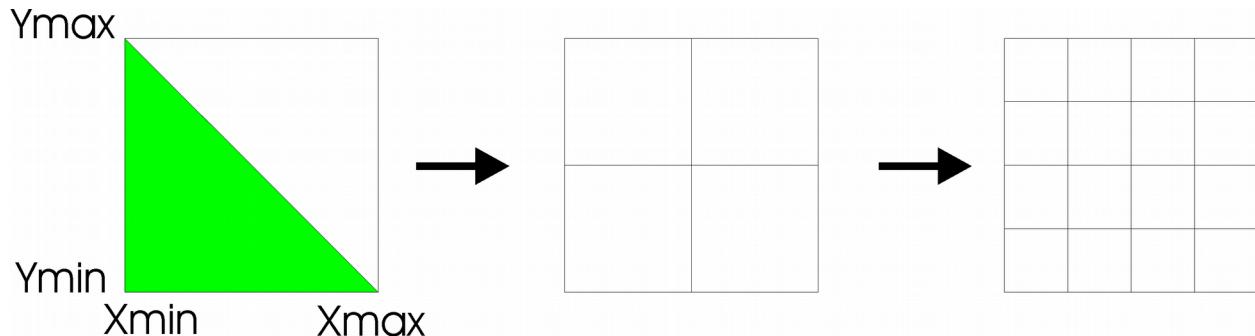


- 2) Generar el grafo de escena incluyendo las transformaciones, tal que partiendo de un cubo, un cilindro y un cono unidad, se pueda realizar el modelo de una cámara de TV situada en el borde de una plataforma de radio  $r$ . La plataforma (cilindro) rota con respecto al eje Y, la cámara rota toda ella con respecto a su base (cilindro), y permite subir y bajar, y el cuerpo de la cámara (cubo) y el objetivo (cono) rotan arriba-abajo e izquierda-derecha. Las medidas de las partes del modelo se dejan a discreción. Implementar el modelo en C++ y OpenGL. (nota: se han dibujado los ejes cartesianos como cilindros y no forman parte del modelo). (2.5 pt)
- 
- A 3D perspective diagram showing a camera setup. A yellow cube representing the camera body sits on top of a grey cylindrical platform. A green vertical cylinder represents the camera lens. The platform rotates around a vertical axis. A horizontal dashed red line represents the ground plane. A blue line extends from the camera lens towards the viewer, representing the optical axis. Dashed black arrows on the platform indicate rotation around the Y-axis. A double-headed arrow on the platform indicates height adjustment. A radius
- $r$
- is shown from the center of the platform to its edge.

- 3) Dada la siguiente imagen que representa simultáneamente el planeta tierra de día y de noche, escriba mediante código C++ la asignación de texturas para una hora  $h$  del día a la malla poligonal que representa una esfera `Esfera::Esfera(int nmeridianos, int nparalelos)` creada con la siguiente llamada `Esfera=new Esfera(24,24)`, suponiendo que el eje de rotación de la Tierra permanece en todo momento perpendicular al plano orbital con respecto al sol. (2.5 pt)



- 4) Dado una función que calcula la intersección de una línea con un triángulo `bool line_triangle_intersection(_vertex3f v1, _vertex3f v2, _vertex3f v3, _vertex3f p1, _vertex3f p2, _vertex3f &Result)`, un modelo 3D definido con vértices y caras (`vector<_vertex3f> Vertices; vector<_vertex3i> Triangulos`), y siendo el tamaño de la imagen MxN píxeles, implementar el pseudocódigo que realiza un ***pick***, devolviendo el índice del triángulo intersectado más cercano al observador. (2.5 pt)



1) Solución: en cada nivel se divide el espacio por la mitad.

```
recursivo(int Nivel,float Xmin,float Xmax,float Ymin,float Ymax, int
Color)
{
if (Nivel==1) dibujar_triangulo(Xmin,Ymin,Xmax,Ymin,Xmin,Ymax, Color);
else{
    float Xmedio=(Xmin+Xmax)/2;
    float Ymedio=(Ymin+Ymax)/2;
    recursivo(Nivel-1,Xmin,Xmedio,Ymin,Ymedio,1);
    recursivo(Nivel-1,Xmin,Xmedio,Ymedio,Ymax,2);
    recursivo(Nivel-1,Xmedio,Xmax,Ymedio,Ymax,3);
}
}
```

2) Ver gráfico para ejemplo. El código se haría en función del grafo. Habría que ajustar los parámetros.

Errores muy graves (0): No poner jerarquía. Sólo tener un nivel.

3) Ver gráfico para entender la idea. Error: ¡Las coordenadas de textura siempre van entre 0 y 1! Para entenderlo, hacer el ejemplo con una malla de 2x2 divisiones

4) Error (0): usar pick de OpenGL. Pseudocódigo

```
Pasar la posición (x,y) a coordenadas de mundo (X,Y)
Si (Proyeccion_perspectiva) P0=(X,Y,Plano_delantero),
P1=Centro_proyección=(0,0,0)

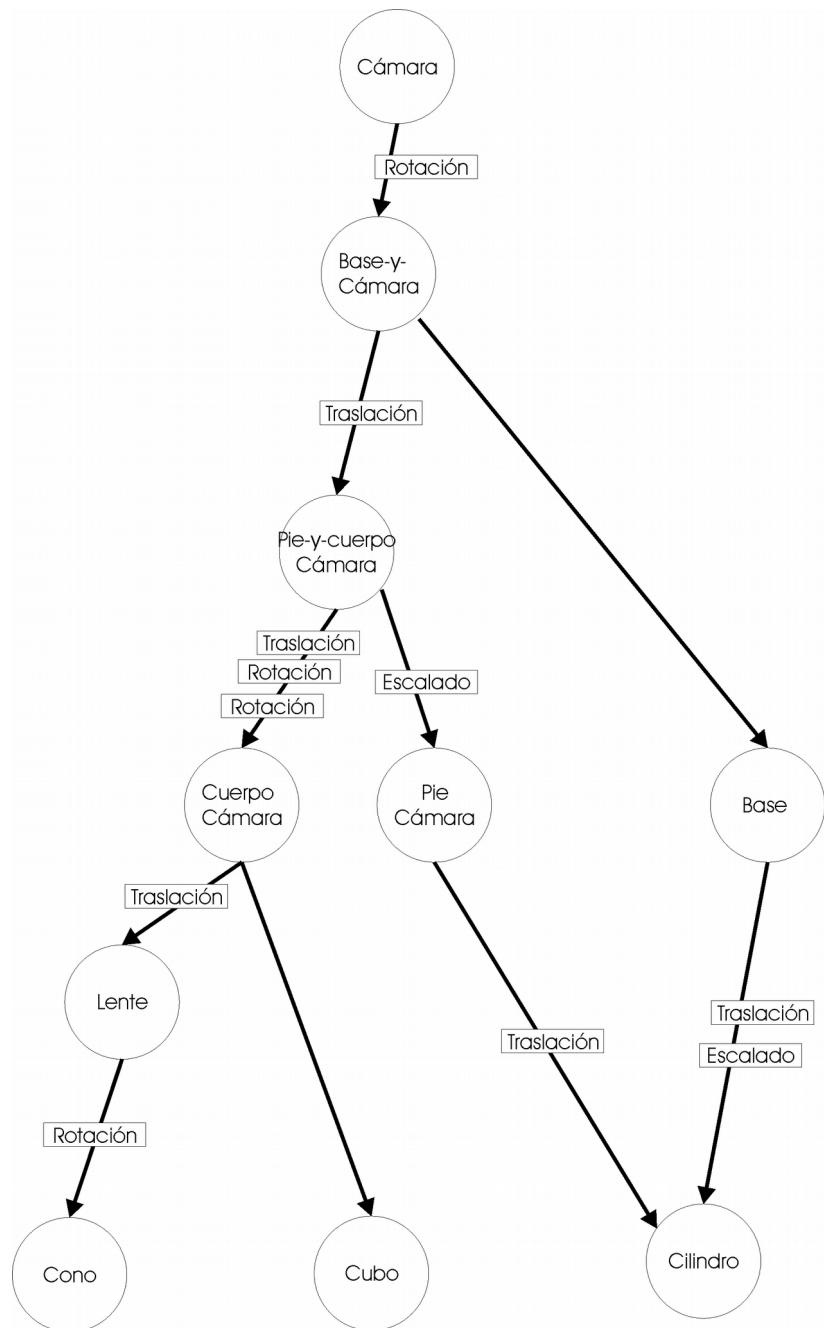
Sino P0=(X,Y,Plano_delantero), P1=(X,Y,0);

Zmin=Infinito;
Posicion=-1;
```

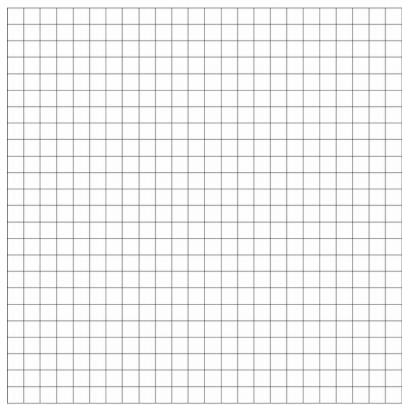
```

Para (i=0, i<triangulos.size(), i++) {
    triangulo=triangulos[i];
    Si (interseccion_linea_triangulo(linea,triangulo, resultado)==true) {
        Si (Resultado.z<zmin) {
            Zmin=Resultado.z;
            Posicion=i;
        }
    }
}
}

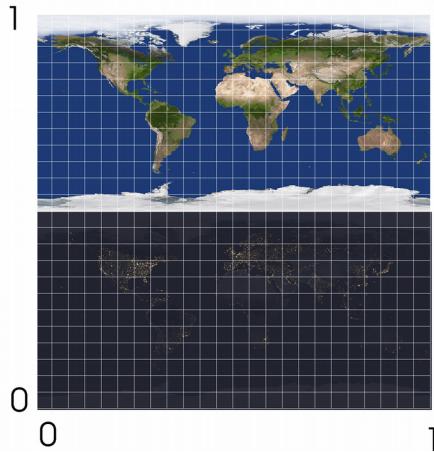
```



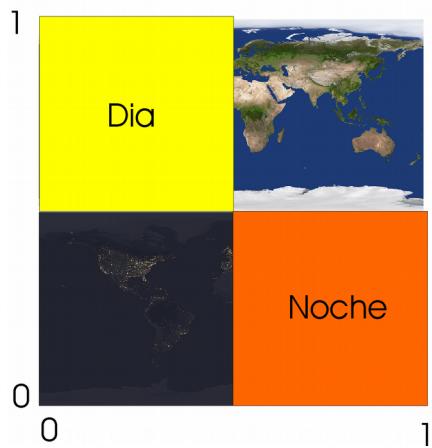
Malla



Textura

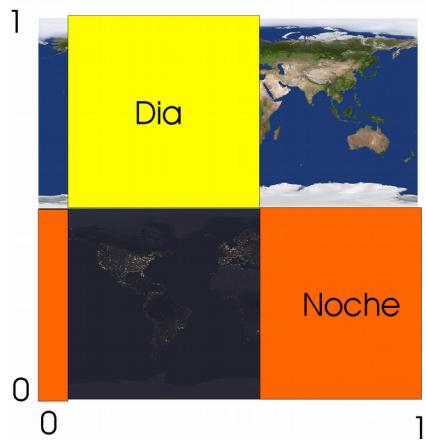


Textura

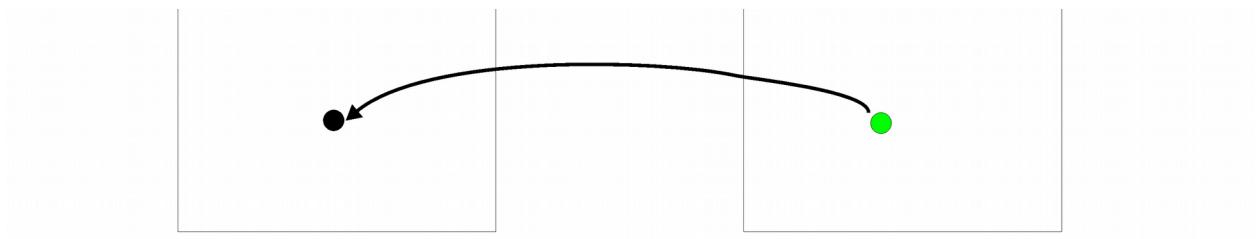


Dia      Noche

Textura

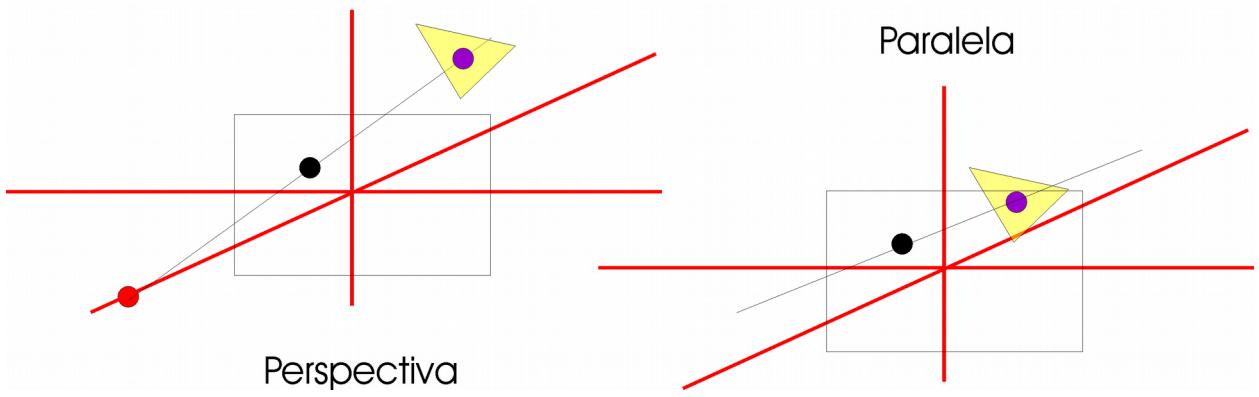


Dia      Noche



Mundo

Dispositivo



Perspectiva

Paralela