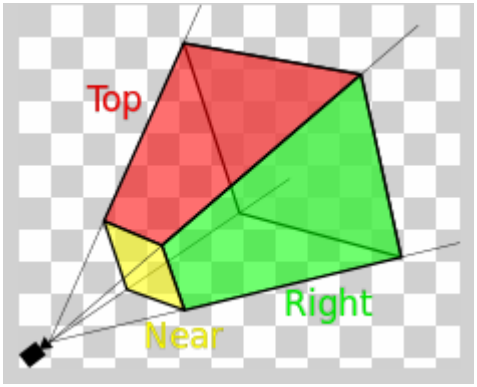
Frustum culling y quadtrees

P. J. Martín, A. Gavilanes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática
Universidad Complutense de Madrid

Frustum Culling

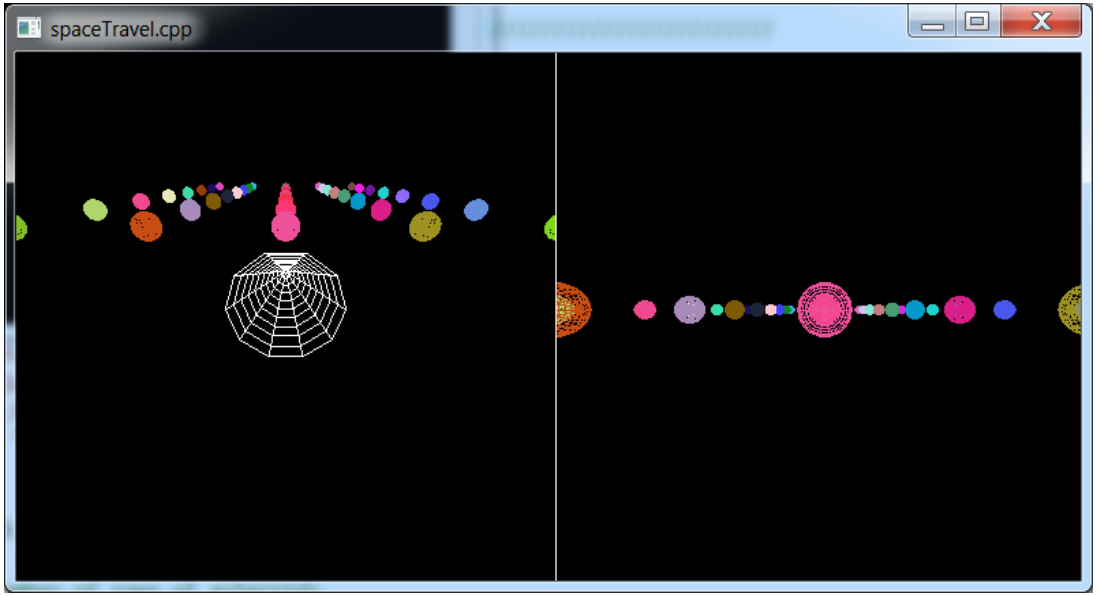
- *Frustum:* pirámide truncada que representa el volumen de vista en la proyección perspectiva.



- Ley de la informática gráfica*: "No pintes lo que no se ve*".
- *(View) Frustum culling*: eliminación de caras de una malla que están fuera del *frustum.*
- El *frustum culling* opera pues a nivel de conjuntos de caras.
- *Culling* de una cara versus *culling* de un conjunto de caras.

Ejemplo

- Los asteroides son esferas con posición, radio y color.
- La escena inicial contiene 8×5 asteroides.
- Una nave (cono blanco) se mueve entre ellos y no se produce demora en la renderización.



Asteroides

```
class Asteroid {
  public:
      ...
      Asteroid(float x, float y, float z, float r,
         char colorR, char colorG, char colorB);
      void draw();

  private:
      float centerX, centerY, centerZ, radius;
      unsigned char color[3];
};
```

Asteroides

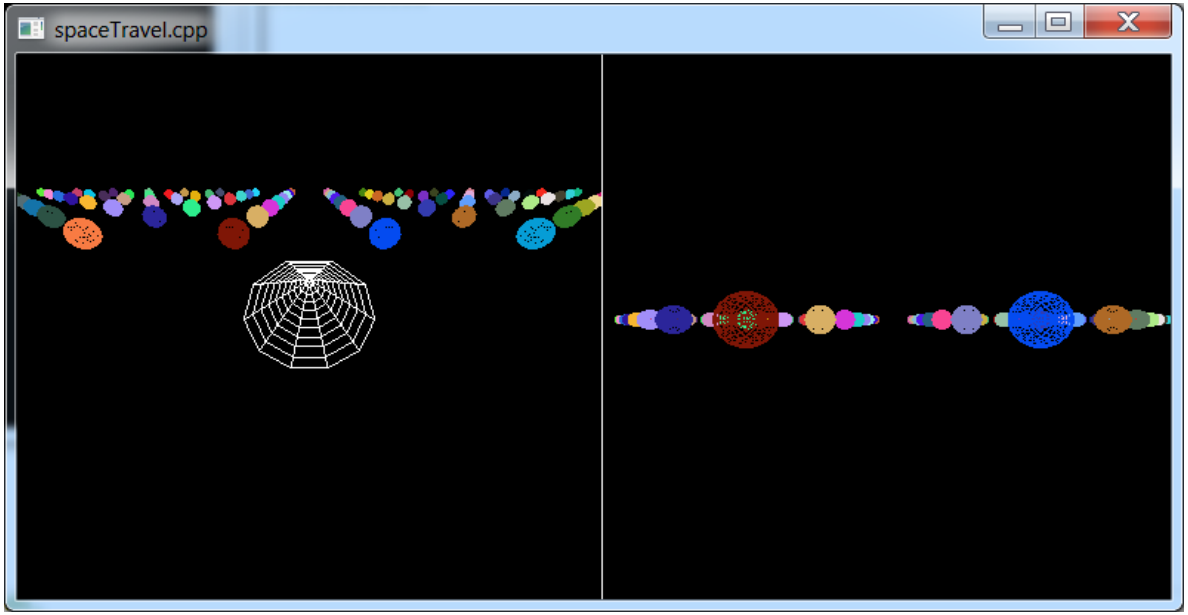
```
void Asteroid::draw() {
  if (radius > 0.0) //If asteroid exists.
  {
     glPushMatrix();
     glTranslatef(centerX, centerY, centerZ);
     glColor3ubv(color);
     glutWireSphere(radius, (int)radius*6,
          (int)radius*6);
     glPopMatrix();
  }
}
```

Ejemplo

```
void drawScene(void) {
   ...
   //Fixed camera.
   gluLookAt(...);
   //Draw all the asteroids in arrayAsteroids.
   for (j=0; j<COLUMNS; j++)
   for (i=0; i<ROWS; i++)
        arrayAsteroids[i][j].draw();
   //Draw spacecraft.
   ...
   //Locate the camera at the tip of the cone
   gluLookAt(...);
   //Draw all the asteroids in arrayAsteroids.
   for (j=0; j<COLUMNS; j++)
   for (i=0; i<ROWS; i++)
        arrayAsteroids[i][j].draw();
}
```

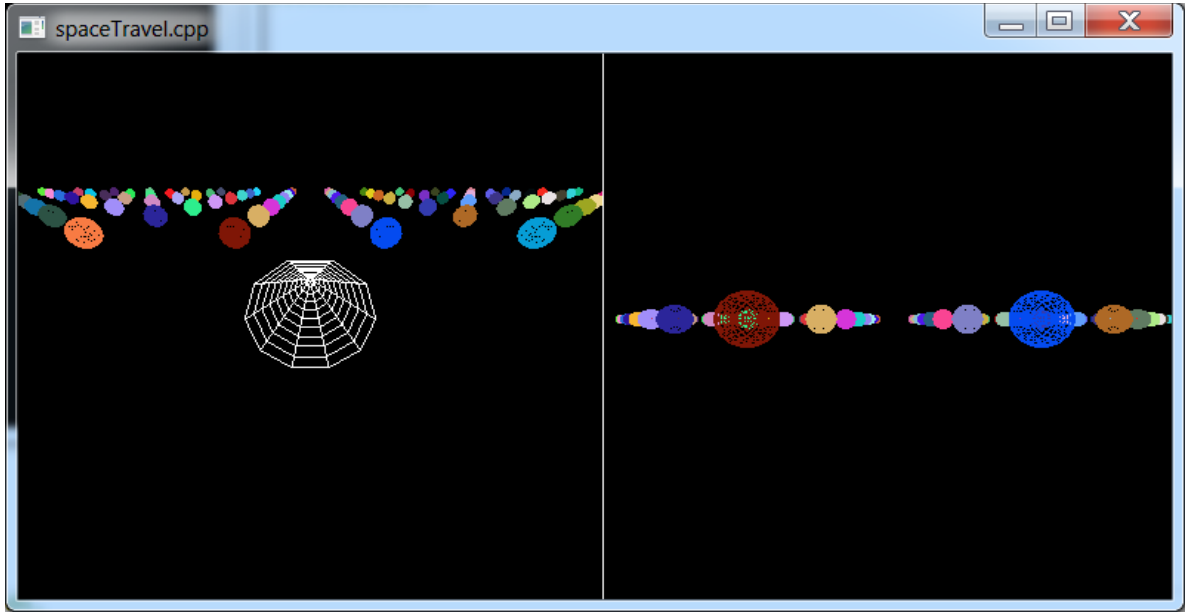
Ejemplo (cont.)

- La escena contiene ahora 100×100 asteroides.
- El movimiento de la nave se demora enormemente.
- El número de asteroides visibles no ha crecido significativamente (del orden de 80 asteroides caben dentro del frustum que se tiene definido).
- ¿Por qué ocurre esto si el renderizado de 80 asteroides no debería ser costoso?



Ejemplo (cont.)

- Explicación:
  - En una rejilla 100×100 se renderizan 10000 asteroides.
  - Cada asteroide tiene un número de caras cuadrático en el radio (`O(radio^2)`).
  - Se debe aplicar la matriz de modelado-vista, la de proyección, hacer clipping de lo que está fuera del volumen de vista y proyectar lo que finalmente ha quedado dentro, para cientos de miles de triángulos.



Frustum culling

- Fin: identificar aquellos objetos que están fuera del frustum, y hacerlo antes de que sean procesados.
- Medios: usar rutinas que permiten chequear si un objeto (o polígono) interseca con el volumen de vista.
- Proceso: filtrar aquellos objetos que no intersecan, guardándolos organizados en estructuras de datos espaciales.

Frustum culling

- Solución naïve: testear cada objeto contra el volumen de vista.
- Solución eficaz: dividir el espacio en celdas que contienen objetos, asumiendo que:
  - La división espacial y la asignación de objetos a celdas solo se hace una vez (muchos objetos estáticos; los dinámicos se pueden dibujar).
  - La división es jerárquica de forma que si una celda no interseca, sus hijos tampoco (poda).
  - El proceso de división de una celda no es costoso (eficiencia en la construcción).
  - El chequeo de intersección entre celda y frustum es sencillo (eficiencia en el uso).

Estructuras espaciales

- La mayoría tiene carácter jerárquico ⟶ **árboles**
  - ✓ Un **nodo interno** tiene asociado un volumen que recubre las primitivas que incluyen sus hijos.
  - ✓ Una **hoja** tiene una colección de (referencias/índices) a primitivas.
- Exploración recursiva
  - ✓ Si el volumen de un nodo interno no es de nuestro interés, todo ese subárbol no se explora (**poda**).
- Construcción
  - ✓ Mediante inserciones de sucesivas primitivas
  - ✓ Algoritmos top-down
  - ✓ Criterios de terminación:
    - Nunca superar una profundidad determinada.
    - Hojas de tamaño limitado.

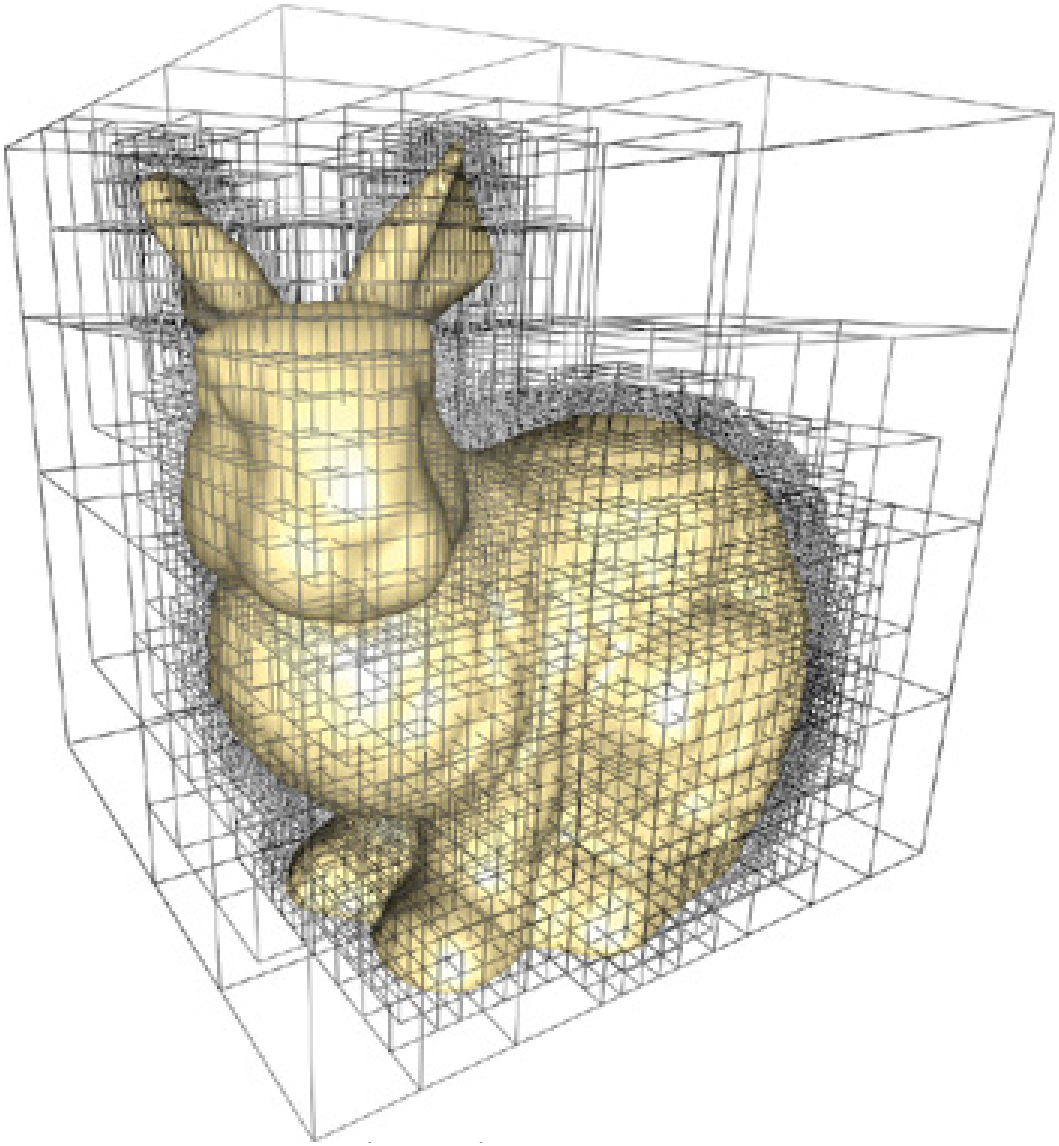
Estructuras espaciales

- Eficiencia
  - ✓ Árboles equilibrados:
    - Algoritmos de construcción sofisticados.
    - Heurísiticas.
  - ✓ Escenas estáticas versus dinámicas:
    - Reconstrucción.
    - Eliminación + inserción.
- Aplicaciones
  - ✓ Ray tracing
  - ✓ Frustum culling
  - ✓ Selección
  - ✓ Colisiones

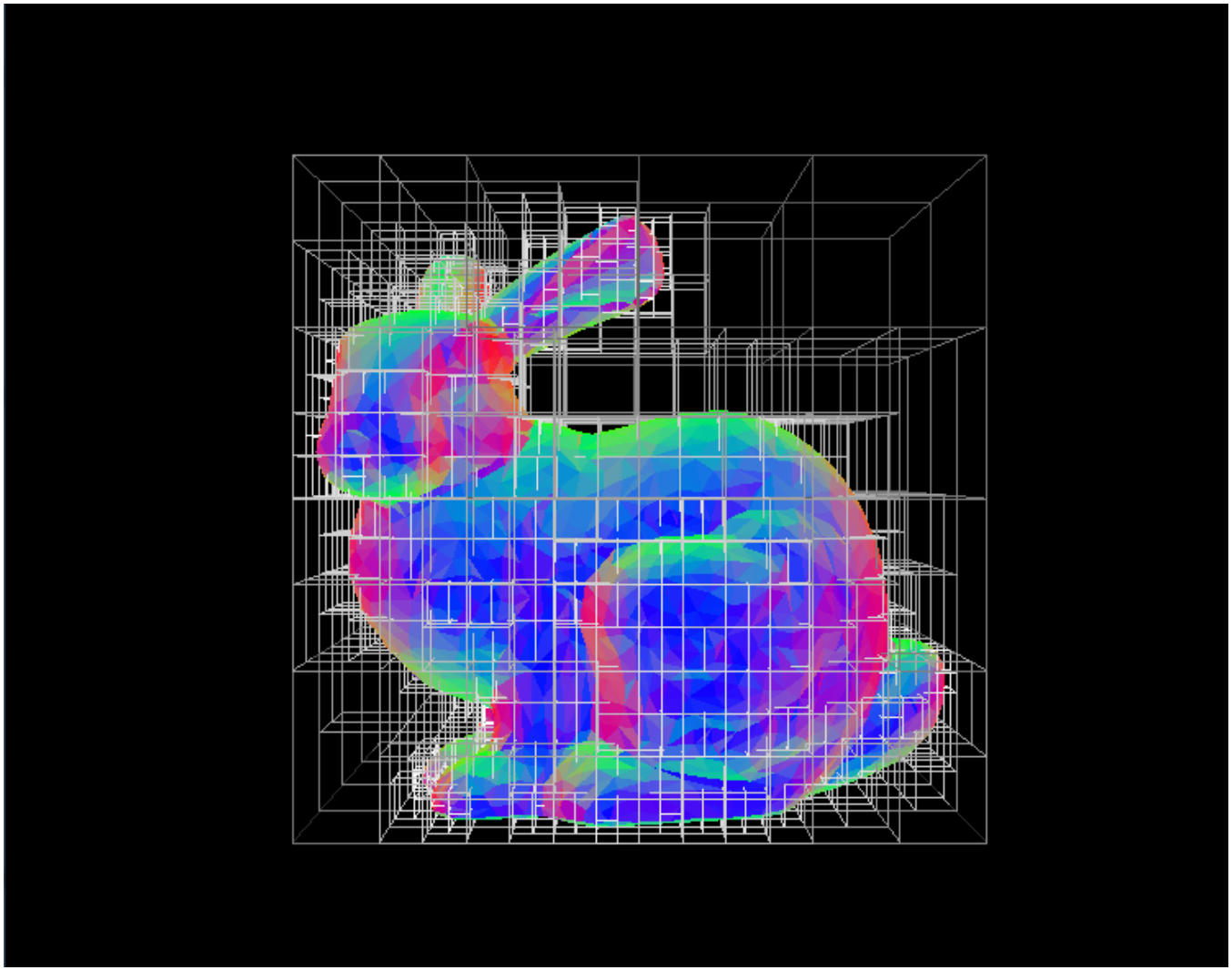
Estructuras espaciales

- **Octrees** (con su versión 2D, quadtrees): partición regular del espacio en cuadrantes.
- **Kd-trees:** partición del espacio por planos paralelos a los formados por los ejes coordenados, tomados dos a dos.
- **Bounding Volume Hierarchies:** división del espacio mediante volúmenes recubridores, que suelen ser esferas o cajas alineadas con los ejes coordenados.
- **Binary Space Partitioning trees:** partición del espacio mediante hiperplanos arbitrarios.

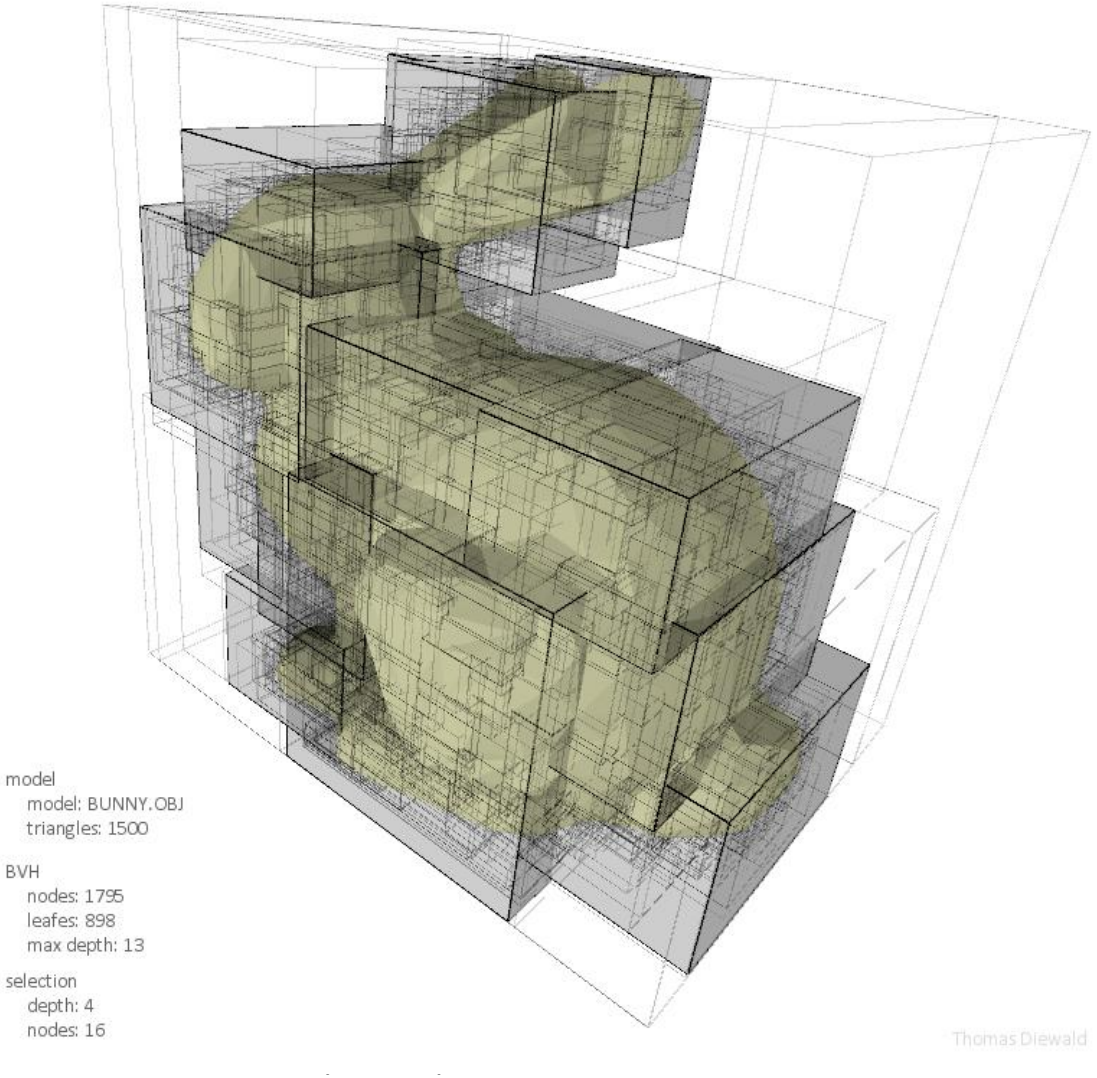
Octree



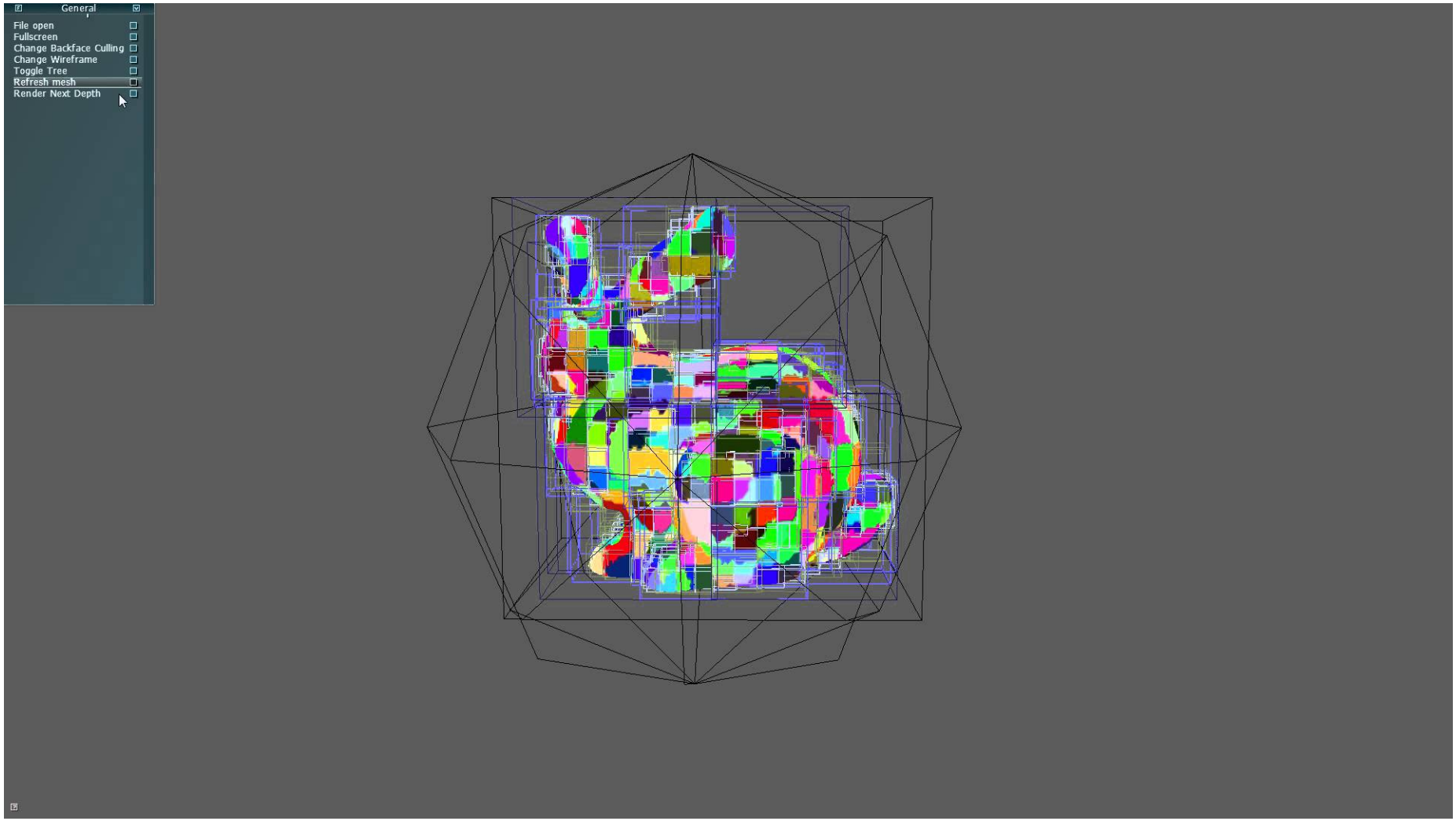
Kd-tree



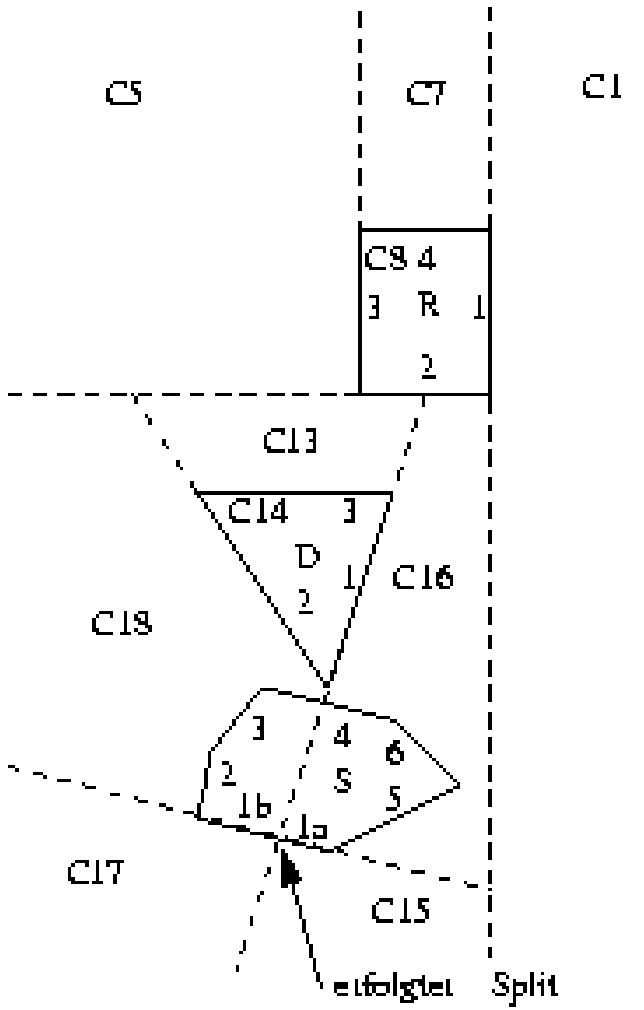
BVH

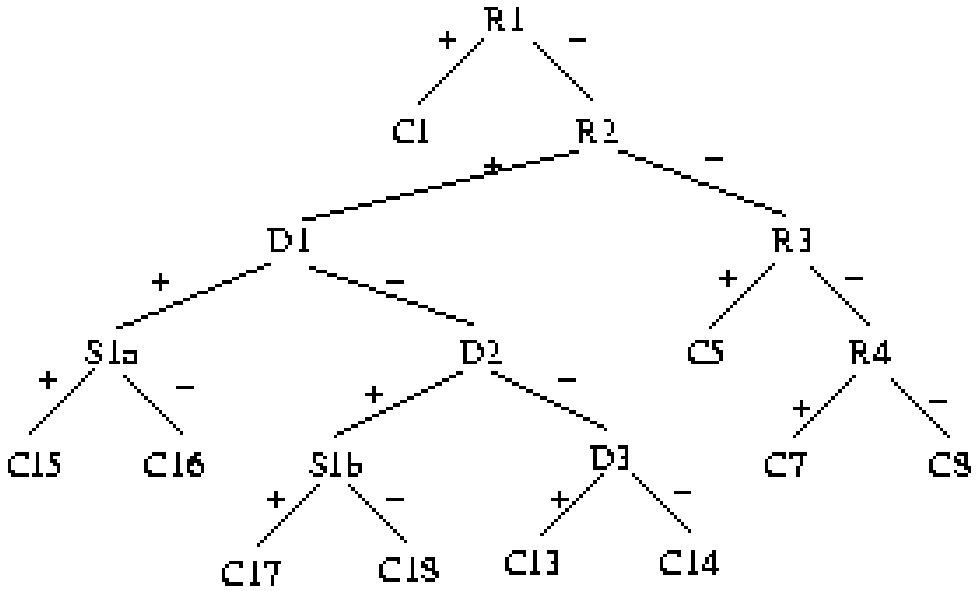

model
model: BUNNY.OBJ
triangles: 1500
BVH
nodes: 1795
leafes: 898
max depth: 13
selection
depth: 4
nodes: 16
Thomas Diewald

BSP

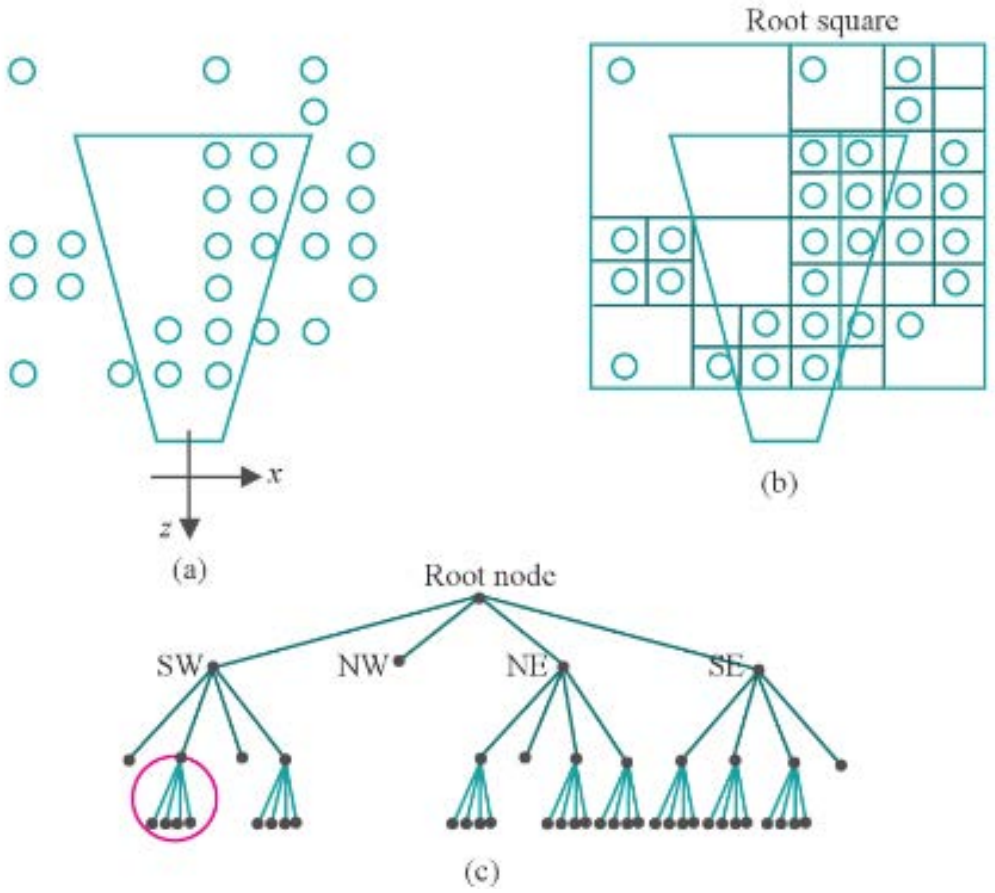

General
File open
Fullscreen
Change Backface Culling
Change Wireframe
Toggle Tree
Refresh mesh
Render Next Depth

BSP


C5
C7
C1
C8
4
3
R
1
2
C13
C14
3
D
1
2
C16
C18
3
4
6
2
S
5
1b
1a
C17
C15
Split


R1
+
−
C1
R2
+
−
D1
R3
+
−
+
−
+
−
S1a
D2
C5
R4
+
−
+
−
+
−
C15
C16
S1b
D3
C7
C8
+
−
+
−
C17
C18
C13
C14

Quadtree ejemplo



Construcción de quadtrees

- Clase `QuadtreeNode`
  - Atributos: **`SWCornerX, SWCornerZ`** (coordenadas de la esquina inferior izquierda), **`size`** (tamaño del cuadrado), **`asteroidList`** (lista de asteroides que intersecan con el nodo; solo para nodos hoja con 1 asteroide, a lo más), **`QuadtreeNode *SWChild, *NWChild, *NEChild, *SEChild`** (punteros a los nodos hijo; no nulos solo para nodos internos)
  - Operaciones:
    - **`build()`**: construye recursivamente el nodo
    - **`numberAsteroidsIntersected()`**: recorre todos los asteroides y, por cada uno, averigua si interseca con el cuadrado asociado al nodo
    - **`addIntersectingAsteroidsToList()`**: construye una lista con todos los asteroides que intersecan con el nodo
    - **`drawAsteroids()`**: dibuja recursivamente los asteroides contenidos en un nodo

Construcción de quadtrees

- **build()**:

```
if (this->numberAsteroidsIntersected() <= 1)
    this->addIntersectingAsteroidsToList();
else {
    SWChild=new QuadtreeNode(SWCornerX,SWCornerZ,size/2.0);
    NWChild=new QuadtreeNode(SWCornerX,SWCornerZ-size/2.0,
           size/2.0);
    NEChild=new QuadtreeNode(SWCornerX+size/2.0,
           SWCornerZ-size/2.0,size/2.0);
    SEChild=new QuadtreeNode(SWCornerX+size/2.0,SWCornerZ,
           size/2.0);
    SWChild->build(); NWChild->build();
    NEChild->build(); SEChild->build();
}
```

Construcción de quadtrees

- **numberAsteroidsIntersected()**:

```
int numVal = 0;
for (int j=0; j<COLUMNS; j++)
for (int i=0; i<ROWS; i++)
  if (arrayAsteroids[i][j].getRadius()>0.0)
  if (checkDiscRectangleIntersection(
          SWCornerX,  SWCornerZ,
          SWCornerX+size, SWCornerZ-size,
          arrayAsteroids[i][j].getCenterX(),
          arrayAsteroids[i][j].getCenterZ(),
          arrayAsteroids[i][j].getRadius()))
    numVal++;
    return numVal;
```

Construcción de quadtrees

- **addIntersectingAsteroidsToList()**:

```
for (int j=0; j<COLUMNS; j++)
for (int i=0; i<ROWS; i++)
    if (arrayAsteroids[i][j].getRadius() > 0.0)
    if (checkDiscRectangleIntersection(
          SWCornerX,  SWCornerZ,
          SWCornerX+size, SWCornerZ-size,
          arrayAsteroids[i][j].getCenterX(),
          arrayAsteroids[i][j].getCenterZ(),
          arrayAsteroids[i][j].getRadius()))
    asteroidList.push_back(
          Asteroid(arrayAsteroids[i][j]));
```

Construcción de quadtrees

- **drawAsteroids()**:

```
//If the square does not intersect the frustum do nothing.
if (checkQuadrilateralsIntersection(x1, z1, x2, z2, x3, z3, x4, z4,
     SWCornerX, SWCornerZ, SWCornerX, SWCornerZ-size,
     SWCornerX+size, SWCornerZ-size, SWCornerX+size, SWCornerZ)) {
         if (SWChild == NULL) { //Square is leaf.
         //Draw all the asteroids in asteroidList.
         ...}
         else {
             SWChild->drawAsteroids(x1, z1, x2, z2, x3, z3, x4, z4);
             NWChild->drawAsteroids(x1, z1, x2, z2, x3, z3, x4, z4);
             NEChild->drawAsteroids(x1, z1, x2, z2, x3, z3, x4, z4);
             SEChild->drawAsteroids(x1, z1, x2, z2, x3, z3, x4, z4);
         }
```

Construcción de quadtrees

- Clase `Quadtree`
  - Atributos: **`QuadtreeNode *header`**
  - Operaciones:
    - **`initialize(float x,float z,float s)`**: construye recursivamente el árbol
    - **`drawAsteroids(float x1,float z1, float x2,float z2, float x3,float z3, float x4,float z4)`**: dibuja los asteroides de la lista de asteroides de aquellos nodos hoja que intersecan con el frustum

Construcción de quadtrees

- **initialize(x, z, s)**:

```
header=new QuadtreeNode(x, z, s);
header->build();
```

- **drawAsteroids(x1,z1,x2,z2,x3,z3,x4,z4)**:

```
header->drawAsteroids(x1, z1, x2, z2,
                      x3, z3, x4, z4);
```

Construcción de quadtrees

- Variables globales

```
Asteroid arrayAsteroids[ROWS][COLUMNS];
    //Global array of asteroids.
Quadtree asteroidsQuadtree;
    //Global quadtree.
```

Construcción de quadtrees

```
drawScene():
    //Begin left viewport.
    ...
    if (!isFrustumCulled) { //Draw all the asteroids in arrayAsteroids.
          for (j=0; j<COLUMNS; j++)
          for (i=0; i<ROWS; i++)
              arrayAsteroids[i][j].draw();}
    else //Draw only asteroids in leaf squares of the quadtree
         //that intersect the fixed frustum with apex at the origin.
         asteroidsQuadtree.drawAsteroids(-5.0,-5.0,-250.0,-250.0,250.0,-250.0,5.0,-5.0);
    //Draw spacecraft.

    //Begin right viewport.
    if (!isFrustumCulled) ...
    else //Draw only asteroids in leaf squares of the quadtree
         //that intersect the frustum "carried" by the spacecraft
         //with apex at its tip and oriented with its axis along the spacecraft's axis.
         asteroidsQuadtree.drawAsteroids(xVal-10*sin((PI/180.0)*angle)-…, …);
```

Occlusion culling

Introducción

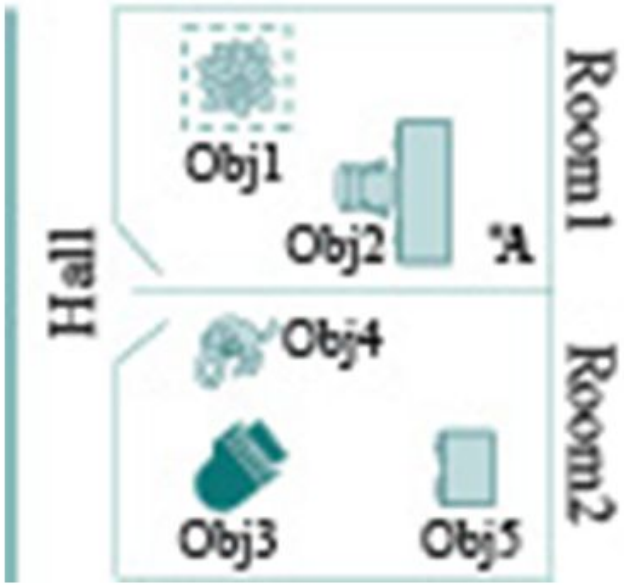


Figure 6.5: Two rooms off a hall. A dashed bounding box is shown containing the first object.

```
if (camera is in Room1) {
        draw Obj1;
        draw Obj2;
}
if (camera is in Room2) {
        draw Obj3;
        draw Obj4;
        draw Obj5;
}
```

Occlusion query

- Occlusion query: consulta para determinar si un objeto es visible, una vez se ha realizado el test de profundidad (por ejemplo, antes de intentar renderizar Obj1, preguntar si la caja que lo contiene es visible)
- Comandos de OpenGL para hacer consultas:
  - `glGenQueries(1, &query)`: genera un objeto consulta `query`.
  - `glBeginQuery(GL_SAMPLES_PASSED, query)`/ `glEndQuery(GL_SAMPLES_PASSED)`: da inicio/finaliza la consulta (es decir, el recuento de fragmentos que pasan el test de profundidad)
  - `glGetQueryObjectuiv(query, GL_QUERY_RESULT, &result)`: devuelve en `result` el número de fragmentos que se han obtenido tras la consulta.

Occlusion query

- **Si el cubo rojo no es visible, "se activa su dibujo", pero sin modificar el buffer de profundidad:**

```
glBeginQuery(GL_SAMPLES_PASSED, query);
//Begin query operation, counting fragments which pass depth test.
if (!boxVisible) {
     glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
            //Color buffer not writable.
     glDepthMask(GL_FALSE);} //Depth buffer not writable.
glutSolidCube(1.0);
if (!boxVisible) {
     glDepthMask(GL_TRUE);  //Depth buffer writable.
     glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);}
            //Color buffer writable.
glEndQuery(GL_SAMPLES_PASSED);
```

- **Si ha habido fragmentos visibles del cubo rojo, se pinta la esfera:**

```
if (result) glutWireSphere(0.5, 16, 10);
```