Sprite ordering and camera culling

By Pau Pedra

Index

- Camera Culling

  - Frustum culling
  - Backface culling
  - Occlusion culling

-Sprite ordering

  - Layers
  - Dynamic sorting

Camera Culling

Why do we need camera culling?

- There will be a bigger limit to the size of the map/levels in the game
- There will be more processing power to be spent on gameplay
- Game will run smoothly in machines with less resources
- Highest resolution and LOD (Level Of Detail) will be more easily achieved
- It’s optimal!

Frustum Culling

In 3D:

-Discard objects outside view area to dramatically reduce draw calls.

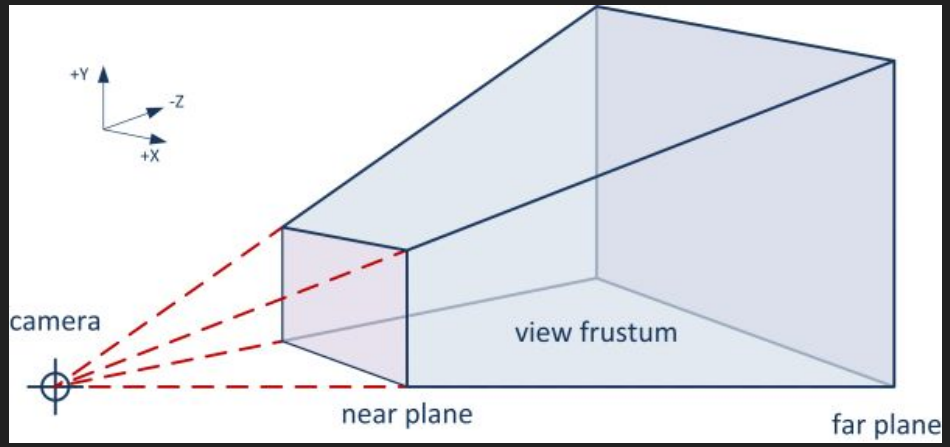
(Horizon Zero Dawn: 2017)

Frustum Culling

In 3D:

-Two planes are created from the view's closest point of view to the furthest

-Truncated pyramid representing view



Frustum Culling

In 2D:

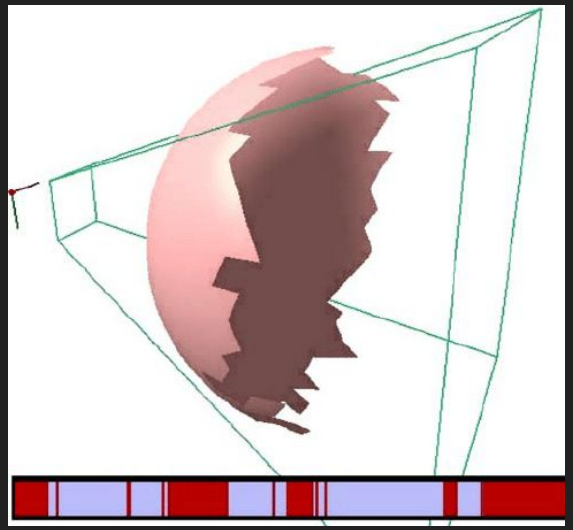
-We have to only draw objects inside screen

-But don’t use Brute Force!! Or at least not always...

Backface Culling

In 3D:

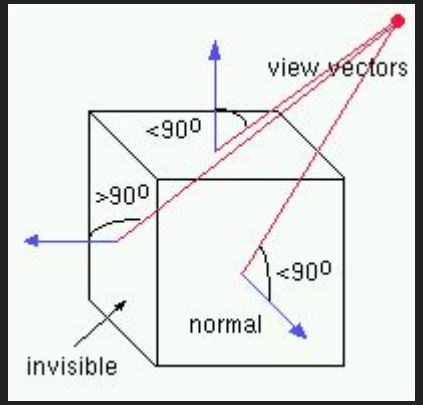
-Discard polygons which won’t be visible



Backface Culling

In 3D:

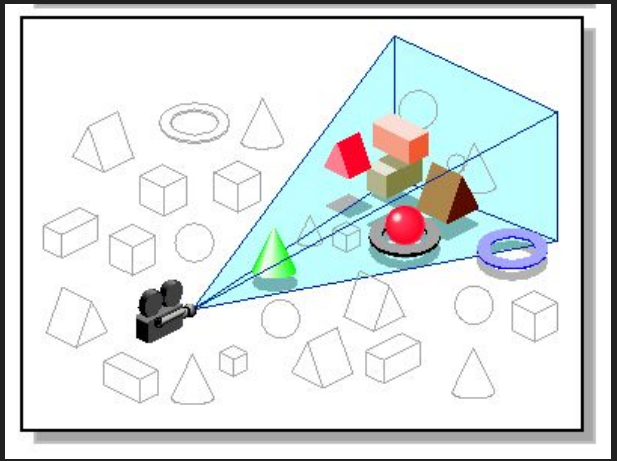
-Calculate dot product between the viewport's vector and the polygon's normal vector



Occlusion Culling

In 3D:

-Discard pixels which will be occluded by other objects in view

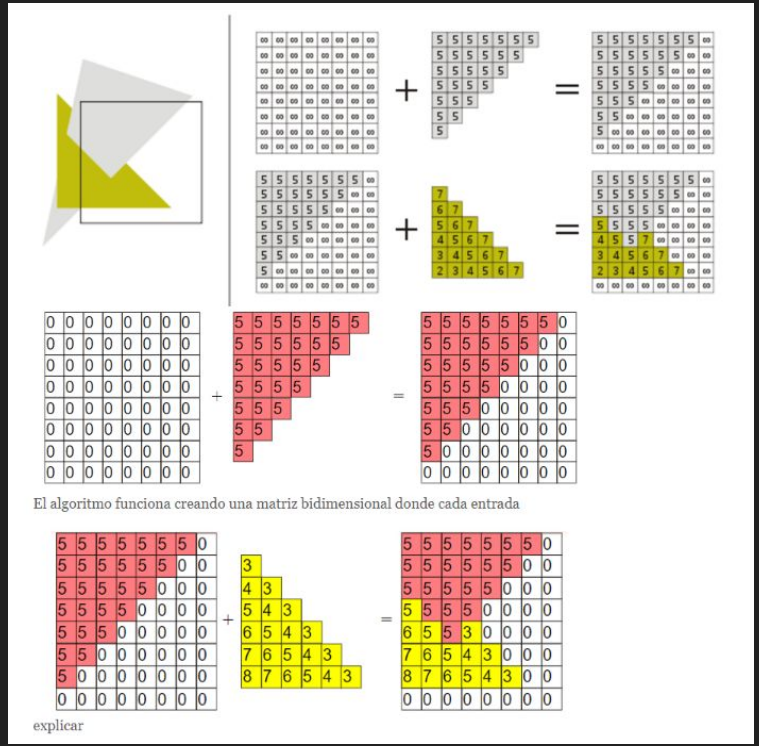


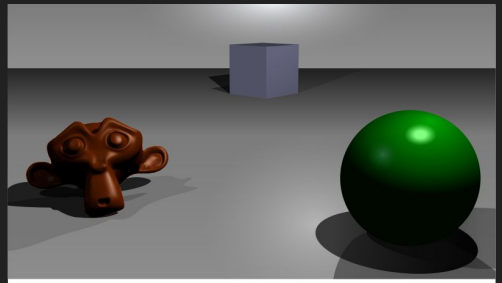
Occlusion Culling

In 3D:

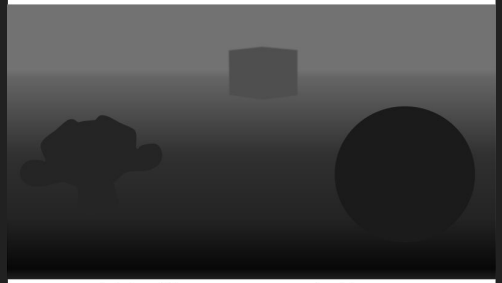
- Z-Buffering

-Painter’s algorithm





A simple three-dimensional scene



Z-buffer representation

Occlusion culling

In 2D:

-Avoid drawing occluded objects

-Not really applicable to all 2D games

-Try and find some ways of doing this in your games!

Sprite ordering

Some theory:

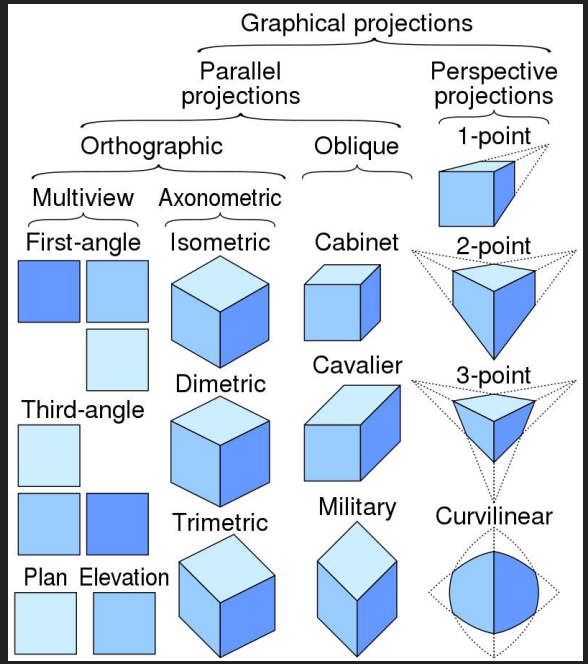
-Orthographic:

Multiview

Axonometric

-Oblique:

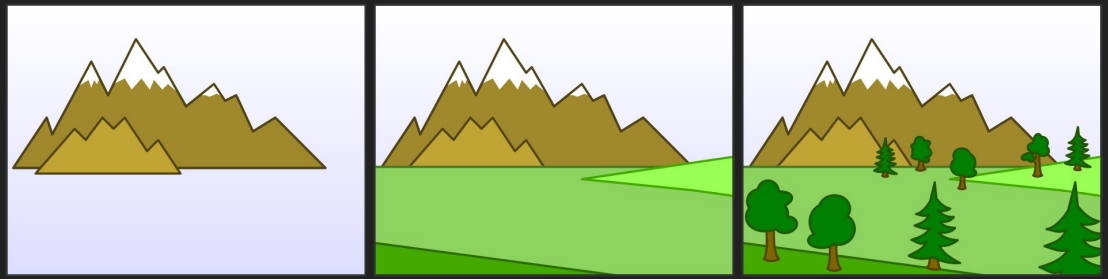
Not perspective but almost



Sprite ordering

Basic concepts:

-Objects are drawn from farthest to closest



-Remember painter’s algorithm…?

Sprite ordering

Multiview:

-2 dimensions

-Static Layers

Example:

-Background->enemies->player->platforms

(Metroid fusion: 2002)

Sprite ordering

Axonometric or Oblique:

-2D but represents 3D

-Static Layers

Example:

-Background->enemies->player->platforms

(Final fantasy Tactics: 2003)

Sprite ordering

Perspective theory:

-The higher the base of an object is on the world the furthest it is from the viewer

-Objects further away are smaller

-Closer objects overlap furthest objects

(Final fantasy Tactics: 2003)

Sprite ordering

Techniques found:

-Smart Layering

-Cheating

(Legend of Zelda: A link to the past 1991)

Sprite ordering

Techniques found:

-But it works!

(Legend of Zelda: A link to the past 1991)

Sprite ordering

Techniques found:

-Not always...

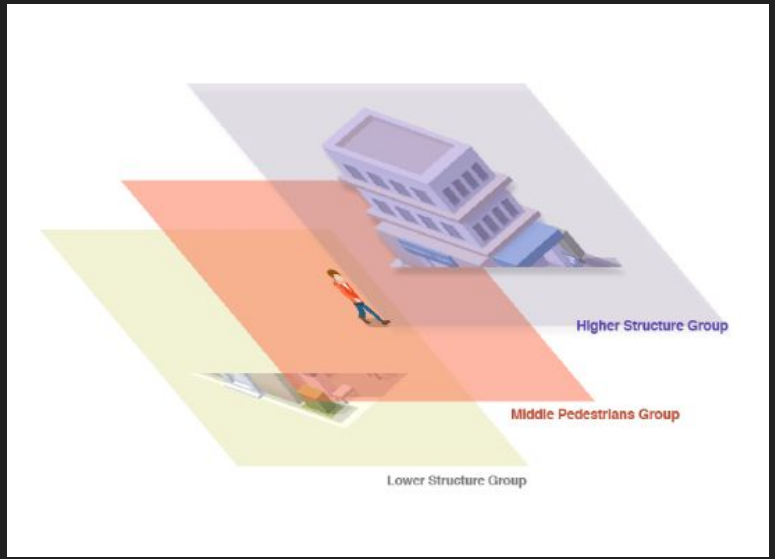
(Legend of Zelda: A link to the past 1991)

Sprite ordering

Techniques found:

-Cutting sprites and layering

-Also cheating



(Pocket CIty 1991)

Sprite ordering

Techniques found:

-But it works!

(Pocket CIty 1991)

Sprite ordering

Techniques found:

-Dynamic ordering using Y position

(Chrono trigger 1995)

Chosen approach

Camera Culling:

-Space partition

Sprite Ordering:

-Ordering using Y position

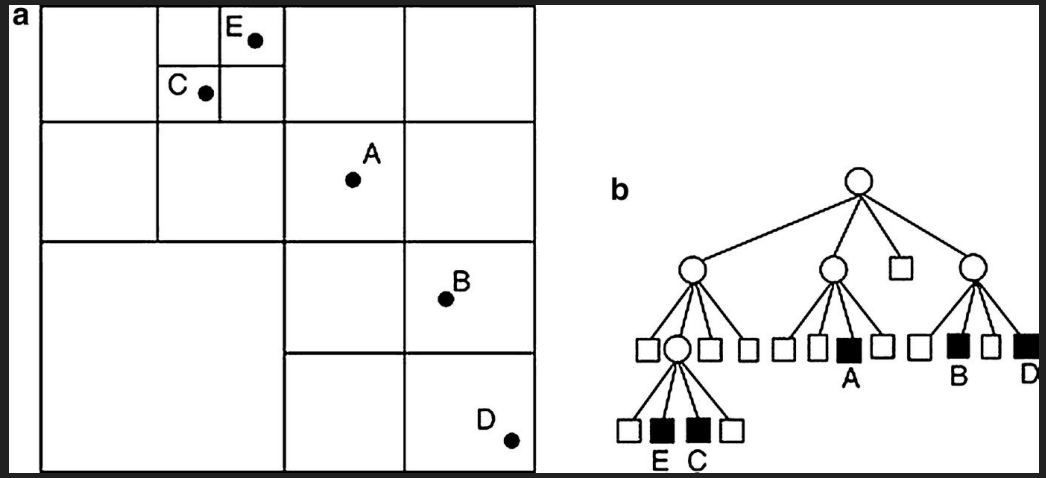
Space partitioning

Quadtrees:

-Tree data structure

-Efficiently cull

-Avoid brute force



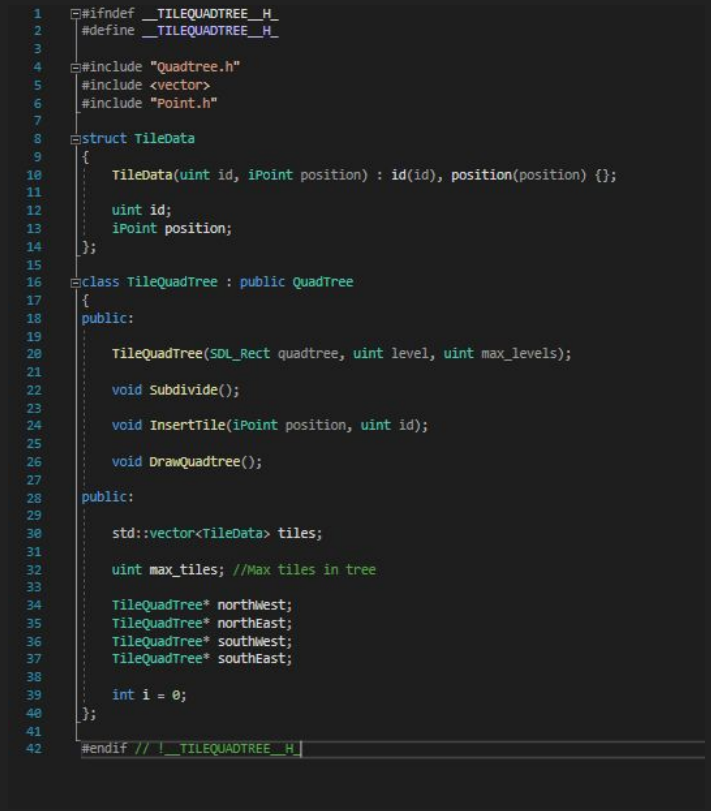
Quadtree

Basics:

-4 subnodes

-Insert

-Subdivide



```
#ifndef __TILEQUADTREE__H_
#define __TILEQUADTREE__H_

#include "Quadtree.h"
#include <vector>
#include "Point.h"

struct TileData
{
    TileData(uint id, iPoint position) : id(id), position(position) {};

    uint id;
    iPoint position;
};

class TileQuadTree : public QuadTree
{
public:

    TileQuadTree(SDL_Rect quadtree, uint level, uint max_levels);

    void Subdivide();

    void InsertTile(iPoint position, uint id);

    void DrawQuadtree();

public:

    std::vector<TileData> tiles;

    uint max_tiles; //Max tiles in tree

    TileQuadTree* northWest;
    TileQuadTree* northEast;
    TileQuadTree* southWest;
    TileQuadTree* southEast;

    int i = 0;
};

#endif // !__TILEQUADTREE__H_
```

Quadtree

Basics:

-Insert

-Subdivide

```
19
20  void TileQuadTree::Subdivide()
21  {
22      if (level < max_levels && !divided)
23      {
24          northWest = new TileQuadTree(SDL_Rect{ boundary.x , boundary.y , boundary.w / 2 , boundary.h / 2 }, level +1, max_levels);
25          northEast = new TileQuadTree(SDL_Rect{ boundary.x + boundary.w / 2 , boundary.y , boundary.w / 2 , boundary.h / 2 }, level +1, max_levels);
26          southWest = new TileQuadTree(SDL_Rect{ boundary.x , boundary.y + boundary.h / 2 , boundary.w / 2 , boundary.h / 2 }, level +1, max_levels);
27          southEast = new TileQuadTree(SDL_Rect{ boundary.x + boundary.w / 2 , boundary.y + boundary.h / 2 , boundary.w / 2 , boundary.h / 2 }, level +1, max_levels);
28
29          divided = true;
30      }
31  }
32
33  void TileQuadTree::InsertTile(iPoint position, uint id)
34  {
35
36      if (!ContainsPoint(position))
37      {
38          return;
39      }
40
41      if(tiles.size() < max_tiles && !divided)
42      {
43          tiles.push_back(TileData(id, position));
44          return;
45      }
46
47      if (level == max_levels)
48      {
49          tiles.push_back(TileData(id, position));
50          return;
51      }
52
53      if (!divided)
54      {
55          Subdivide();
56      }
57
58      northWest->InsertTile(position, id);
59      northEast->InsertTile(position, id);
60      southWest->InsertTile(position, id);
61      southEast->InsertTile(position, id);
62
63  }
```

Quadtree

Basics:

-Query

-Recursively check if rectangle intersects with node's boundaries

-Get objects inside these nodes

```
class QuadTree
{
    ...

    // Find all points that appear within a range
    function queryRange(AABB range)
    {
        // Prepare an array of results
        Array of XY pointsInRange;

        // Automatically abort if the range does not intersect this quad
        if (!boundary.intersectsAABB(range))
            return pointsInRange; // empty list

        // Check objects at this quad level
        for (int p = 0; p < points.size; p++)
        {
            if (range.containsPoint(points[p]))
                pointsInRange.append(points[p]);
        }

        // Terminate here, if there are no children
        if (northWest == null)
            return pointsInRange;

        // Otherwise, add the points from the children
        pointsInRange.appendArray(northWest->queryRange(range));
        pointsInRange.appendArray(northEast->queryRange(range));
        pointsInRange.appendArray(southWest->queryRange(range));
        pointsInRange.appendArray(southEast->queryRange(range));

        return pointsInRange;
    }
}
```

My implementation

Calculations:

-Corners of the screen

-Identify highest and lowest row

-Check if the tile we are drawing is still inside the viewport

```
App->win->GetWindowSize(winWidth, winHeight);                                                    //Gets the w

for (std::list<MapLayer*>::iterator layer = data.layers.begin(); layer != data.layers.end(); layer++)
{
    if (smaller_camera)
    {
        camera_pos_in_pixels.x = -App->render->camera.x + winWidth / 4;
        camera_pos_in_pixels.y = -App->render->camera.y + winHeight / 4;

        bottom_right_x = -App->render->camera.x + winWidth * 0.75;
        bottom_right_y = -App->render->camera.y + winHeight * 0.75;
    }
    else
    {
        camera_pos_in_pixels.x = -App->render->camera.x;
        camera_pos_in_pixels.y = -App->render->camera.y;

        bottom_right_x = -App->render->camera.x + winWidth;
        bottom_right_y = -App->render->camera.y + winHeight;
    }

    min_x_row = WorldToMap(camera_pos_in_pixels.x, camera_pos_in_pixels.y).x;                    //Top Left Corner row
    max_x_row = WorldToMap(bottom_right_x + data.tile_width , bottom_right_y ).x + 1;            //Down Right Corner row

    min_y_row = WorldToMap(bottom_right_x, camera_pos_in_pixels.y).y;                            //Up Righ Corner row
    max_y_row = WorldToMap(camera_pos_in_pixels.x, bottom_right_y + data.tile_height).y + 1;     //Down Left Corner row

    if (min_x_row < 0)
    {
        min_x_row = 0;
    }
    if (min_y_row < 0)
    {
        min_y_row = 0;
    }
```

My implementation: part 2

```
for (int x = min_x_row ; x < max_x_row && x < data.width; x++)
{
    for (int y = min_y_row ; y < max_y_row && y < data.height && MapToWorld(x, y).y < bottom_right_y && MapToWorld(x, y).x > camera_pos_in_pixels.x - data.tile_width; y++)
    {
        if (MapToWorld(x, y).y > camera_pos_in_pixels.y - data.tile_height  && MapToWorld(x, y).x < bottom_right_x)
        {
            int tile_id = (*layer)->Get(x, y);                                          //Gets the tile id from the tile index. Gets the tile index for a

            if (tile_id > 0)
            {
                TileSet* tileset = GetTilesetFromTileId(tile_id);                       //Gets the tileset corresponding with the tile_id. If tile id

                if (tileset != nullptr)
                {
                    SDL_Rect tile_rect = tileset->GetTileRect(tile_id);                 //Gets the position on the world and the dimensions of the re
                    iPoint pos = MapToWorld(x, y);                                      //Gets the position on the world (in pixels) of a specific po

                    App->render->Blit(tileset->texture, pos.x + tileset->offset_x, pos.y + tileset->offset_y, &tile_rect); //, false, (*layer)->speed)

                    tiles_drawn++;
                }
            }
        }
    }
}
```

Entities culling

How it should work:

-Static entities should use quadtrees

-Dynamic entities can use bruteforce if their number is not ridiculously high

Sorting using Y position

Setup:

```
fPoint          center_point;
```

-Add center point to entities

Sorting using Y position

Sorting:

-I used sort() algorithm

-Custom sorting condition

```
entities_in_screen = entities;

std::sort(entities_in_screen.begin(), entities_in_screen.end(), customLess);

DrawEntities();
```

Sorting using Y position

Sorting:

-Struct and condition used

```
struct {
    bool operator()(Entity* a, Entity* b) const
    {
        return a->center_point.y < b->center_point.y;
    }
} customLess;
```

The End