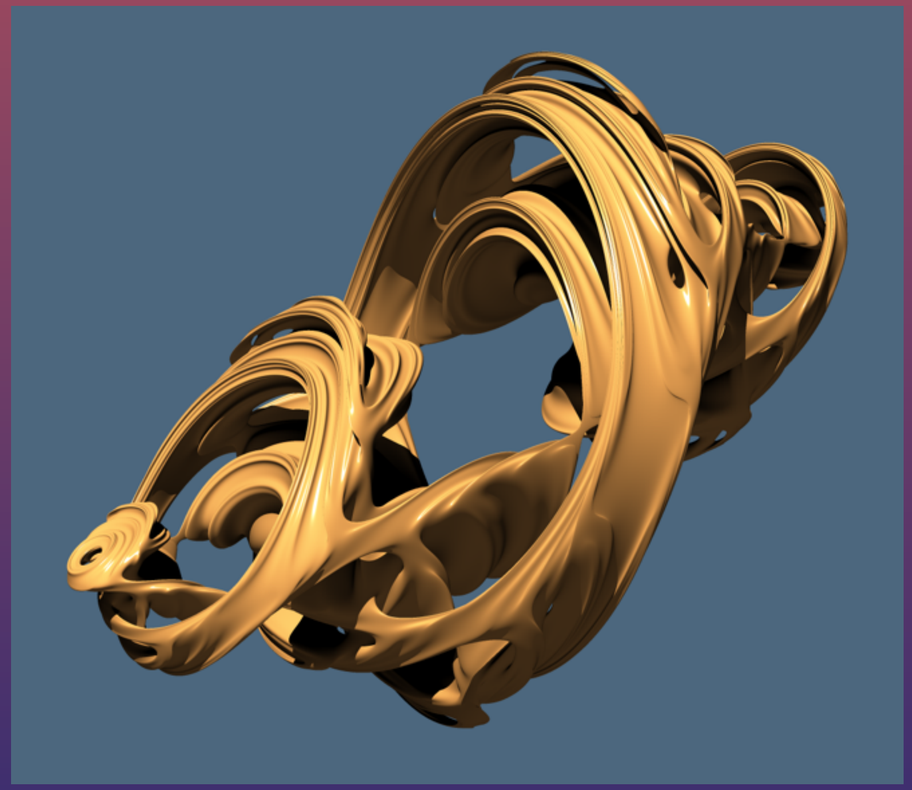
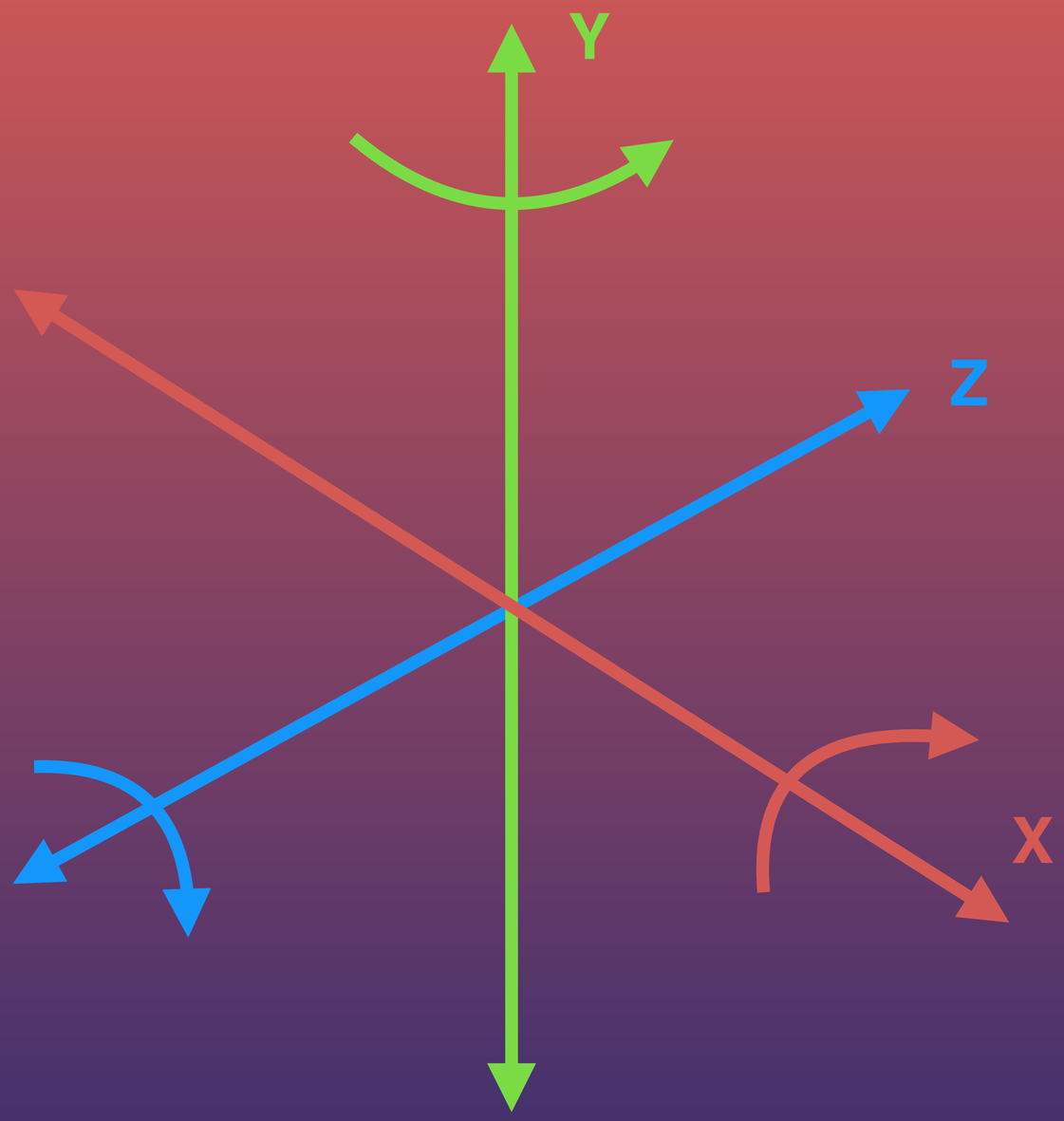
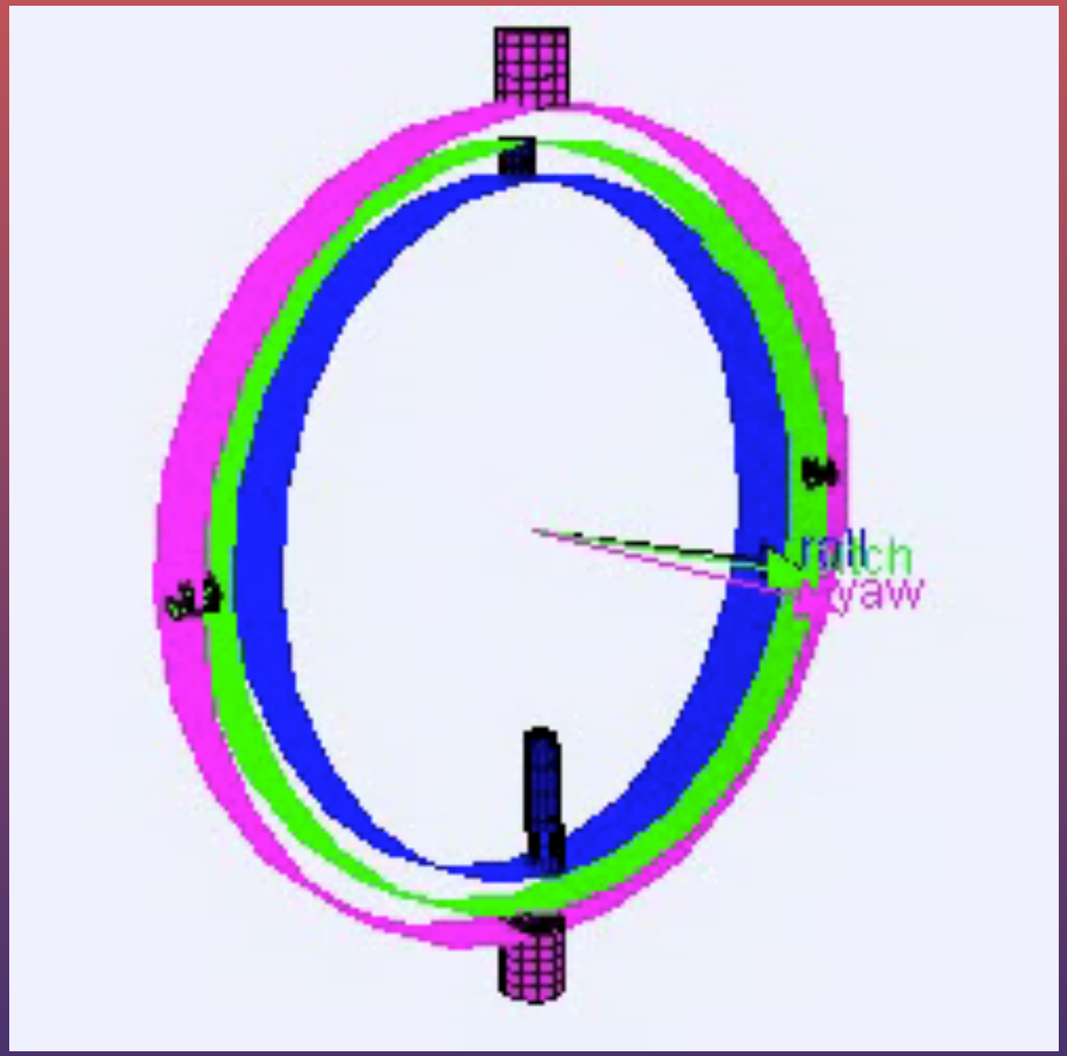
3D / Windowing

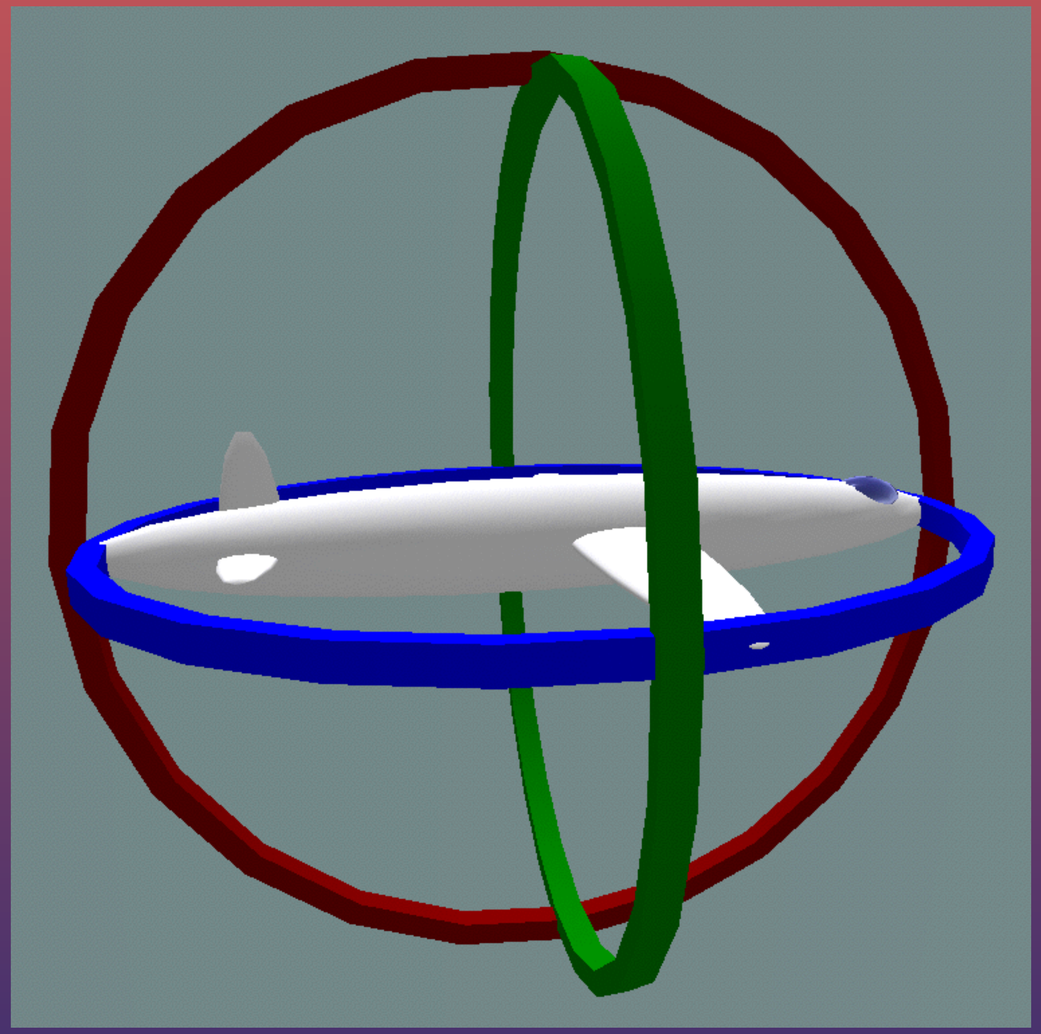


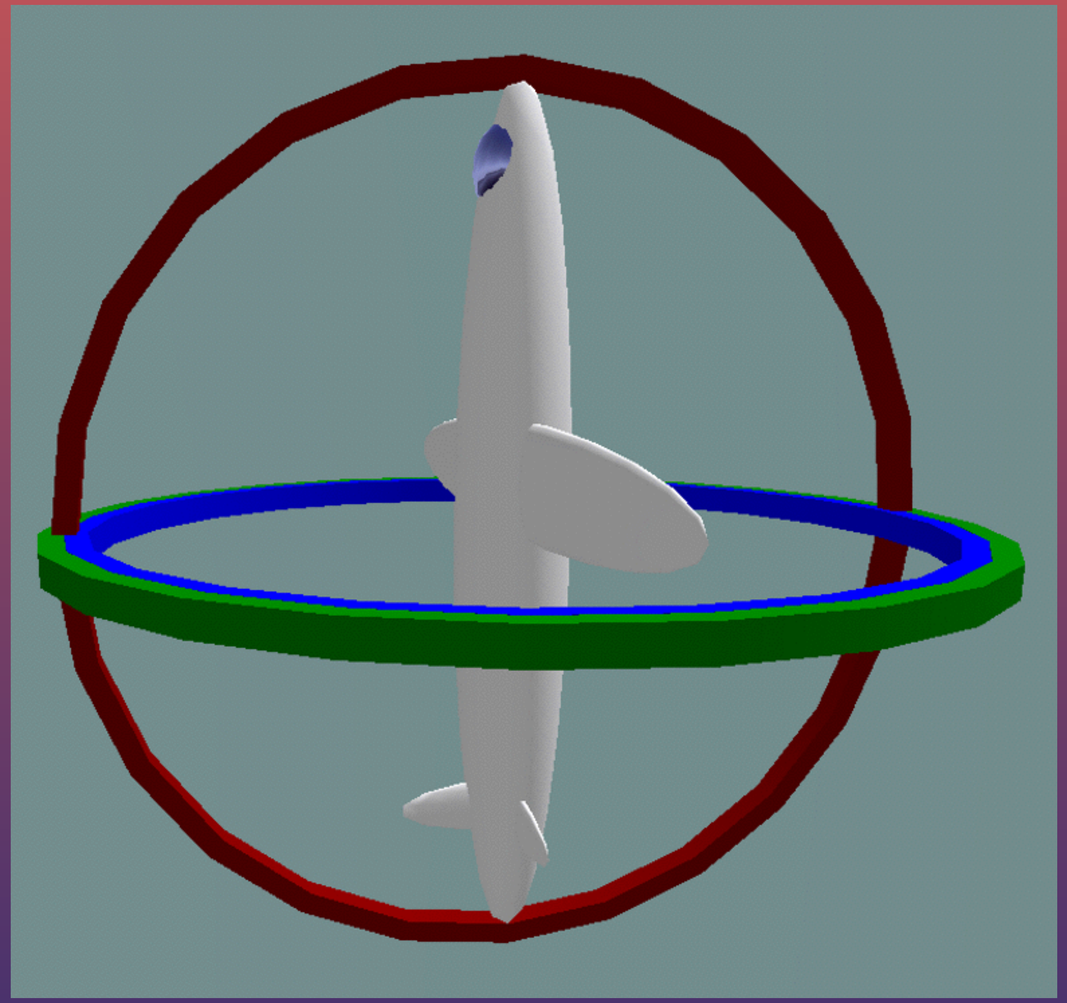
Problems with using Euler angles for rotation.


Y
Z
X

Gimbal lock


roll
pitch
yaw





Need to create 3 separate matrices when creating the final model matrix.

Euler angle rotations are order dependent and thus interpolating each angle independently will give us different rotation paths depending on the angle order.

Quaternions

Quaternion

$$w + xi + yj + zk$$

4-dimensional complex number comprised of real and imaginary numbers.

Storing rotation as a quaternion.

- Allows for 3D rotation interpolation.
- No gimbal lock.
- Can create the full rotation matrix from the quaternion.

Collision detection in 3D.

AABB collision detection is the same in 3D as in 2D,
just need to test one more axis.

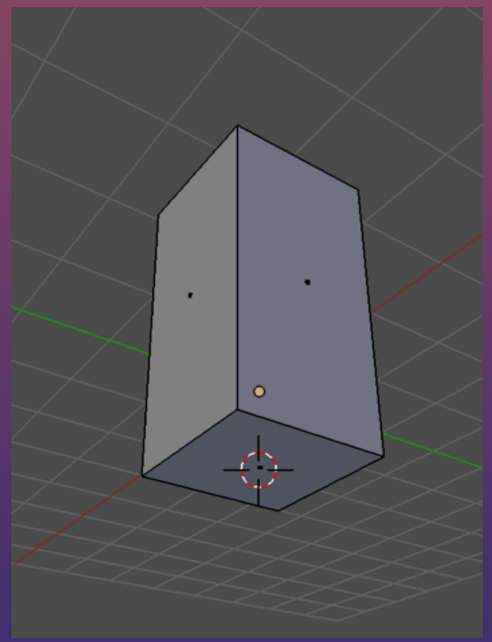
Separate collision into Z, Y and X axis tests like in 2D.

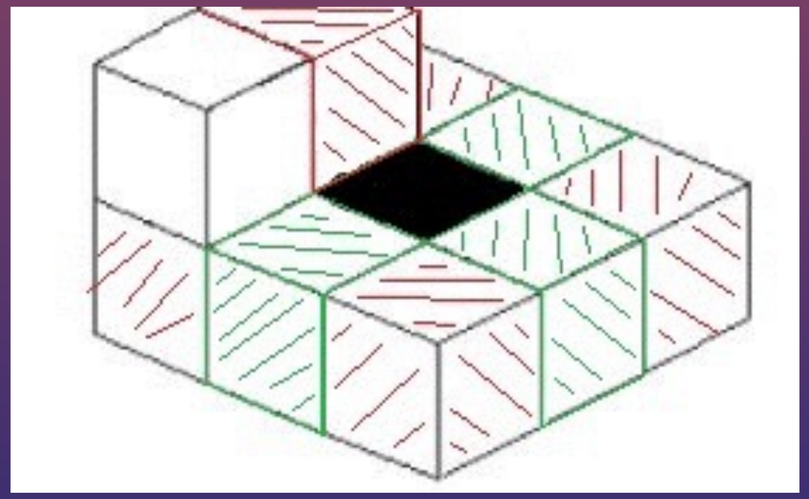
Test bounding CUBES (width, height, depth) instead of RECTANGLES.

Just like in 2D, two cubes are colliding if the distance between them is smaller than (or smaller than or equals to) the sum of their half-dimensions on that axis!

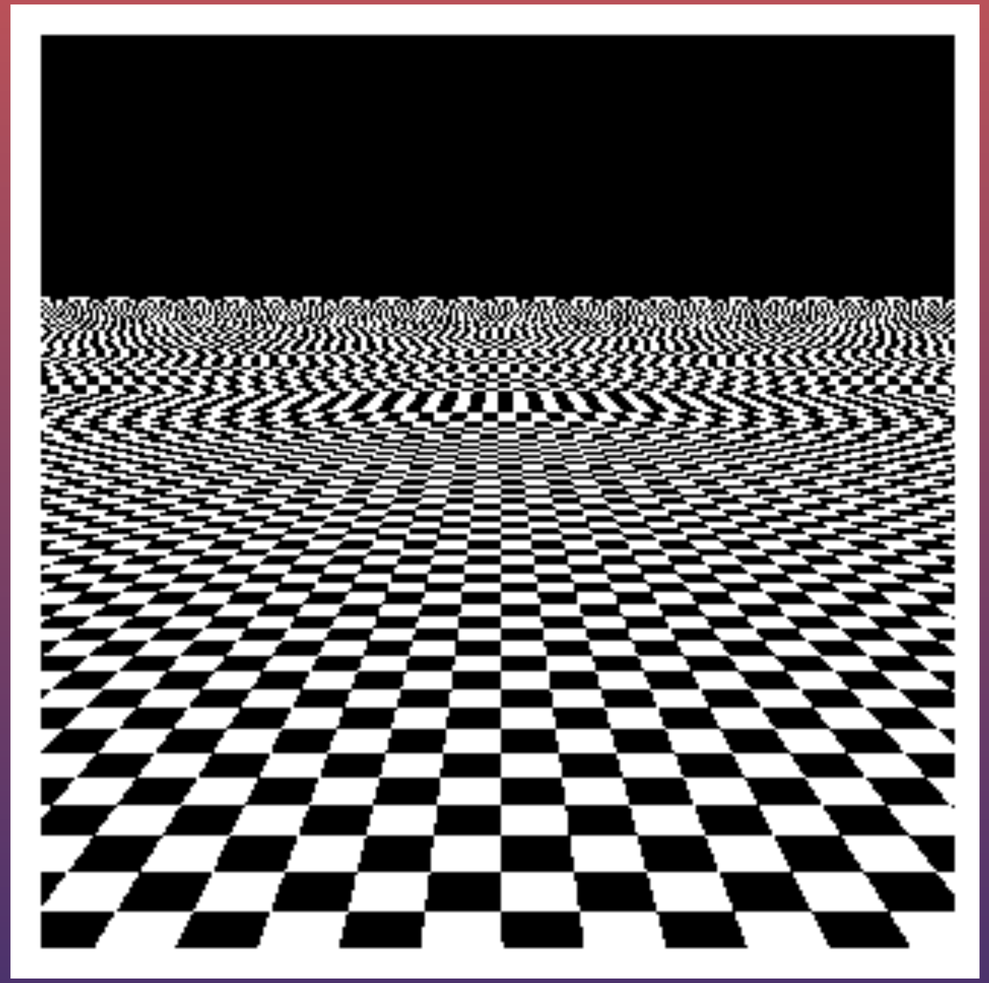
For 3D tilemaps, it’s the same as 2D tilemaps.

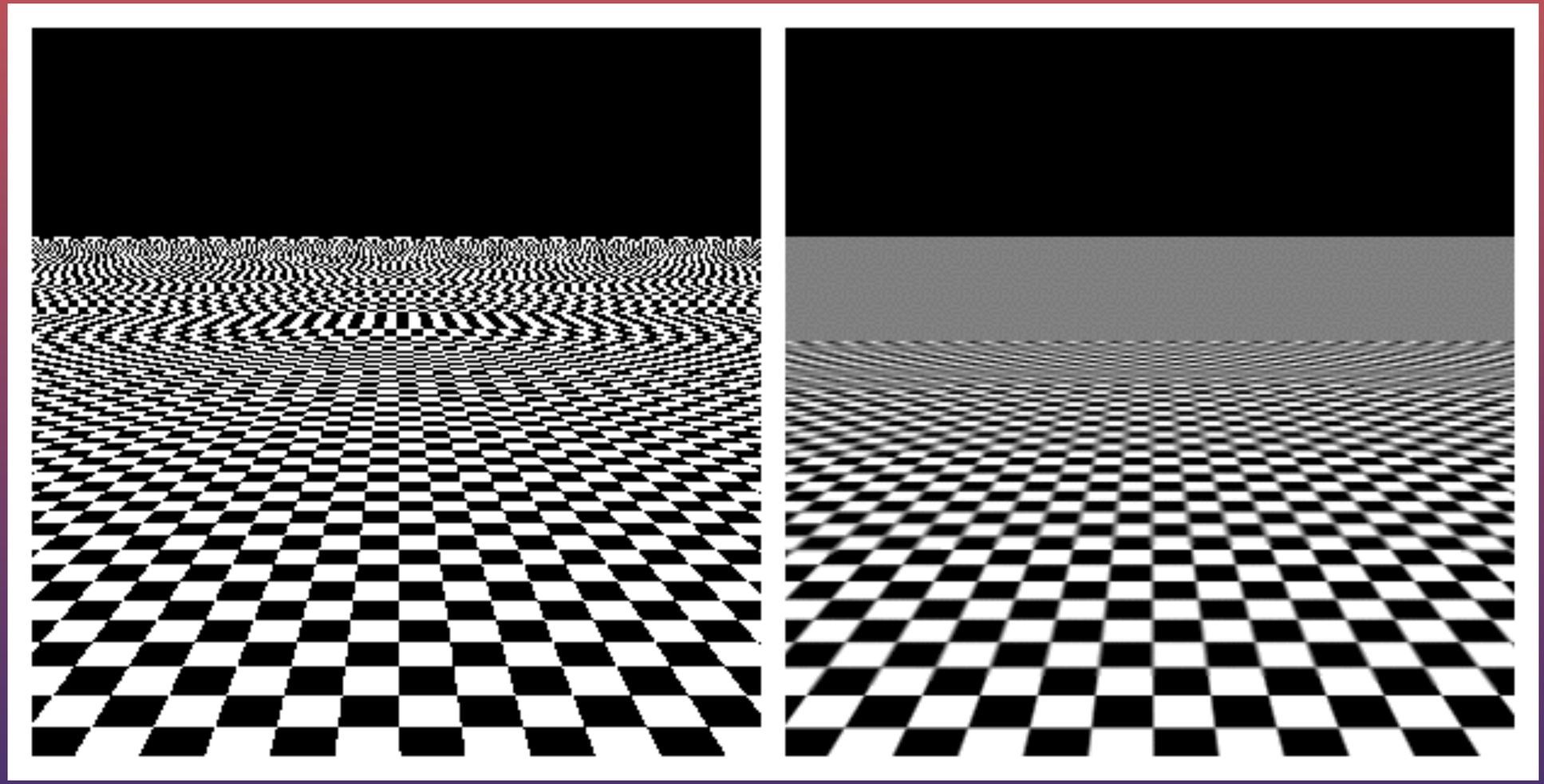
Test points around the player box to see if the tiles they intersect are solid and adjust accordingly.

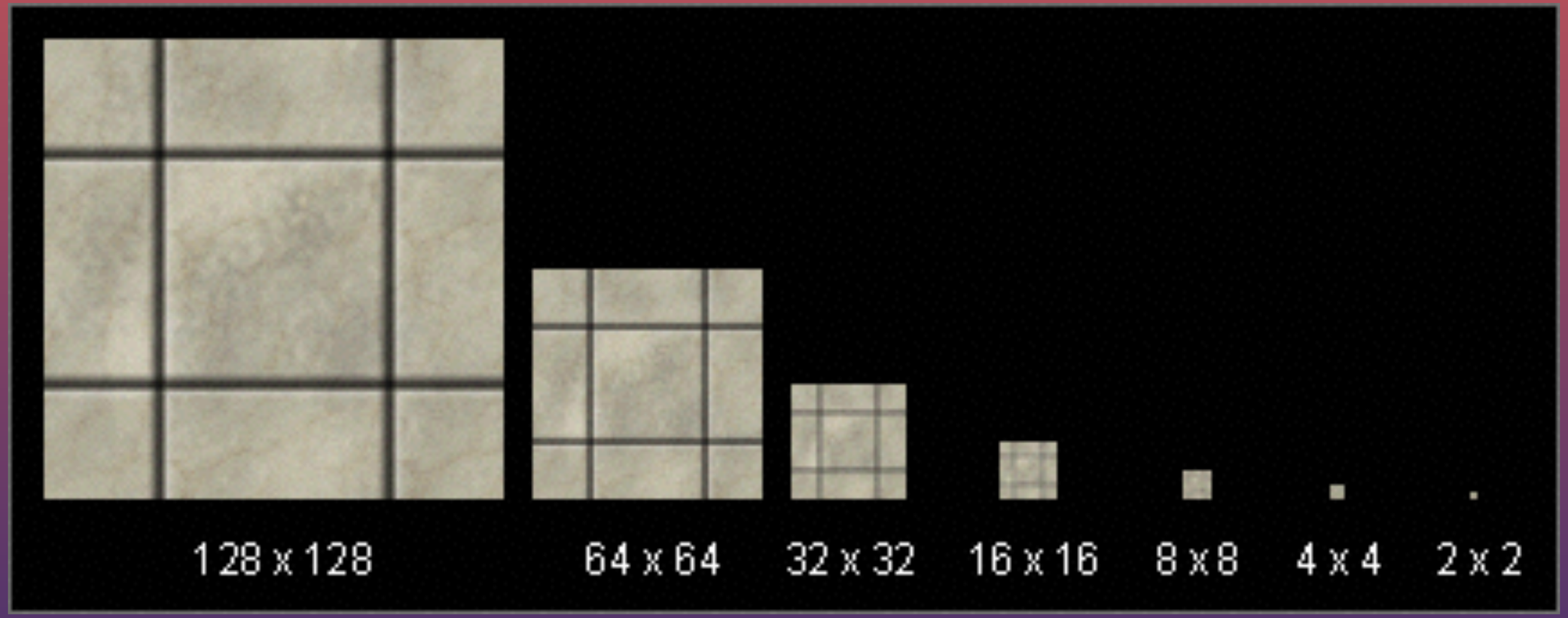




Mipmaps






128 x 128
64 x 64
32 x 32
16 x 16
8 x 8
4 x 4
2 x 2

```
void glTexParameteri (GLenum target, GLenum pname,
GLint param);
```

Sets a texture parameter of the specified texture target.

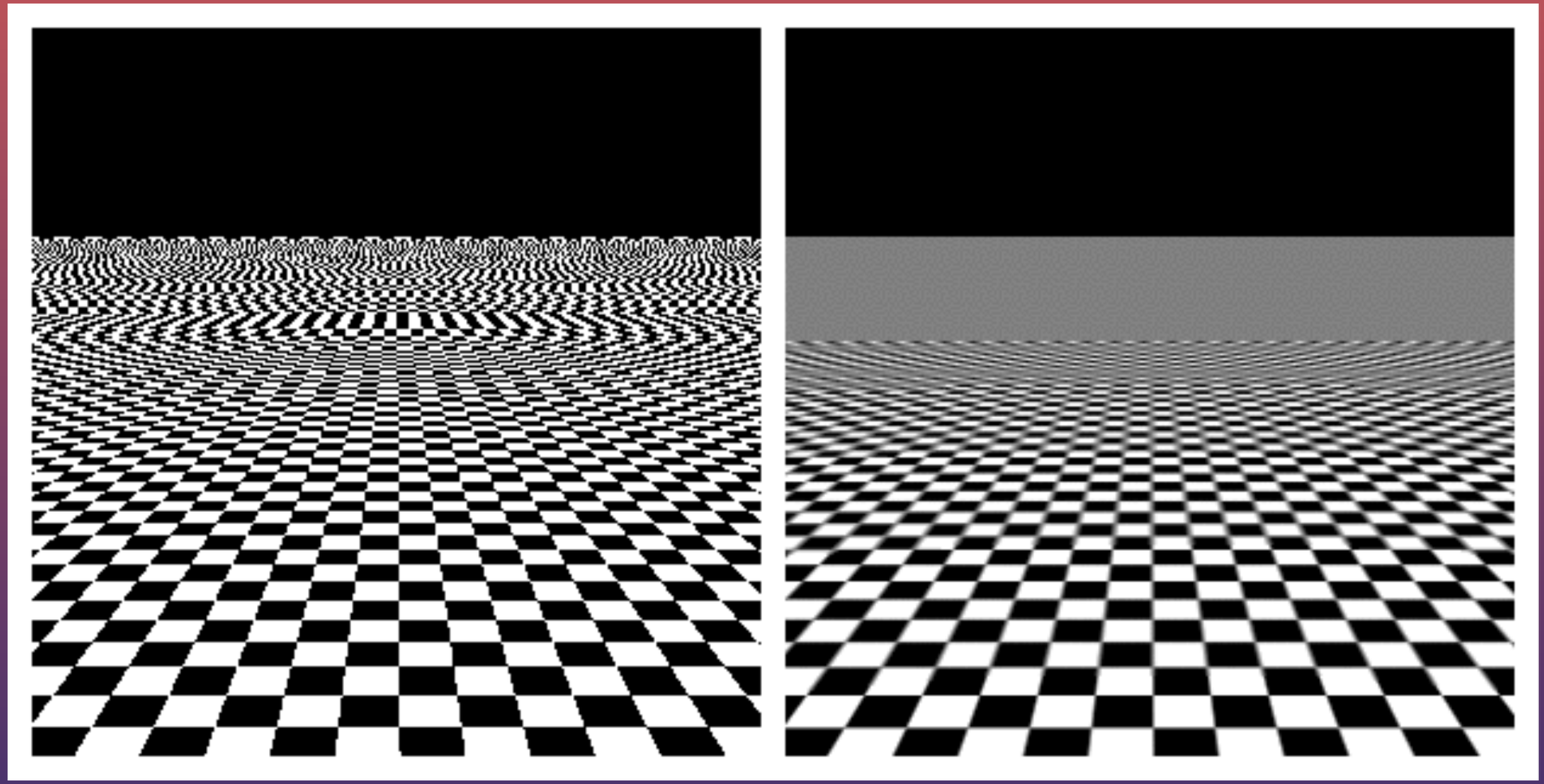
Set **GL_GENERATE_MIPMAP** parameter to **GL_TRUE** to generate mipmaps automatically.

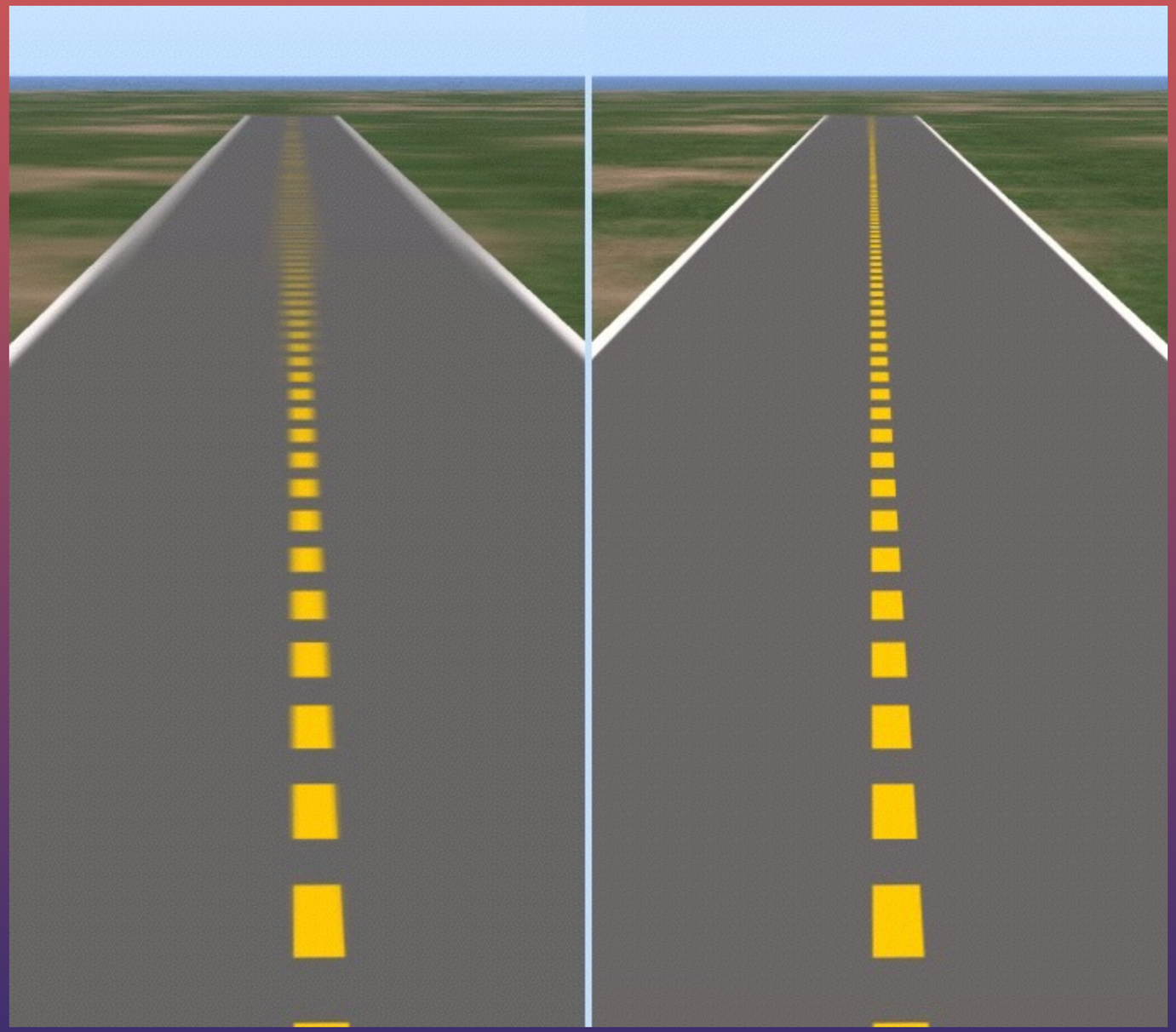
Set **GL_TEXTURE_MIN_FILTER** to **GL_LINEAR_MIPMAP_LINEAR** or **GL_NEAREST_MIPMAP_NEAREST** to set the minification filter to use mipmaps.

```
glTexParameteri(GL_TEXTURE_2D, GL_GENERATE_MIPMAP,GL_TRUE);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
```


114
0
B
Items
A Put away

Anisotropic filtering





```
void glTexParameterf (GLenum target, GLenum pname,
GLfloat param);
```

Sets a floating point texture parameter of the specified texture target.

Set **GL_TEXTURE_MAX_ANISOTROPY_EXT** parameter to set the anisotropic filtering amount. Must be a power of 2 up to 16 (0 (off), 2, 4, 8 or 16).

```
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAX_ANISOTROPY_EXT, 16.0f);
```

Resolutions and fullscreen.

Fullscreen

```
int SDL_SetWindowFullscreen(SDL_Window *window,
Uint32 flags);
```

Sets the specified SDL window to fullscreen or windowed mode. Pass flags as:

SDL_WINDOW_FULLSCREEN or SDL_WINDOW_FULLSCREEN_DESKTOP for fullscreen mode or **0** for windowed mode.

```
SDL_SetWindowFullscreen(displayWindow, SDL_WINDOW_FULLSCREEN); // set
fullscreen

SDL_SetWindowFullscreen(displayWindow, 0); // set windowed mode
```

Enumerating video modes.

`int SDL_GetNumDisplayModes(int displayIndex);`

Returns the number of display modes for given display index (monitor).

```
SDL_GetNumDisplayModes(0); // get number of display modes for main monitor
```

```
int SDL_GetDisplayMode(int displayIndex, int modeIndex, SDL_DisplayMode * mode);
```

Gets the details about a specified mode index and stores it in the SDL_DisplayMode struct pointed to by **mode.**

Use this in conjunction with **SDL_GetNumDisplayModes** to list available resolutions.

```
    for(int i=0; i < SDL_GetNumDisplayModes(0); i++) {
        SDL_DisplayMode mode;
        SDL_GetDisplayMode(0, i, &mode);
        cout << "AVAILABLE RESOLUTION:" << mode.w << "x" << mode.h << endl;
    }
```

Setting a video mode.

In windowed mode.

```
void SDL_SetWindowSize(SDL_Window * window, int w, int h);
```

Sets the window size for a window specified by the SDL_Window pointer.

```
SDL_SetWindowSize(displayWindow, 800, 600); // resize window to 800 x 600
```

In fullscreen

```
int SDL_SetWindowDisplayMode(SDL_Window * window, const SDL_DisplayMode *
mode);
```

Sets the display mode to the mode specified by the **mode** pointer. On some platforms, you need to exit and re-enter fullscreen for the new mode to take effect.

```
SDL_DisplayMode mode;
SDL_GetDisplayMode(0, selectedVideoMode, &mode);

SDL_SetWindowDisplayMode(displayWindow, &mode);
SDL_SetWindowFullscreen(displayWindow, 0);
SDL_SetWindowFullscreen(displayWindow, SDL_WINDOW_FULLSCREEN);
```

Don’t forget to set your viewport and aspect ratios to the new resolution!

```
float aspect = (float)currentResolutionX /(float)currentResolutionY;
setPerspective(65.0f, aspect, 0.1f, 200.0f);


glViewport(0,0,currentResolutionX, currentResolutionY);
glOrtho(-aspect, aspect, -1.0f, 1.0f, -1.0f, 1.0f);
```

Hiding the cursor.

```
int SDL_ShowCursor(int toggle);
```

Show or hide the mouse pointer. Pass **0** to hide and **1** to show.

```
SDL_ShowCursor(0); // hide the pointer
SDL_ShowCursor(1); // show the pointer
```