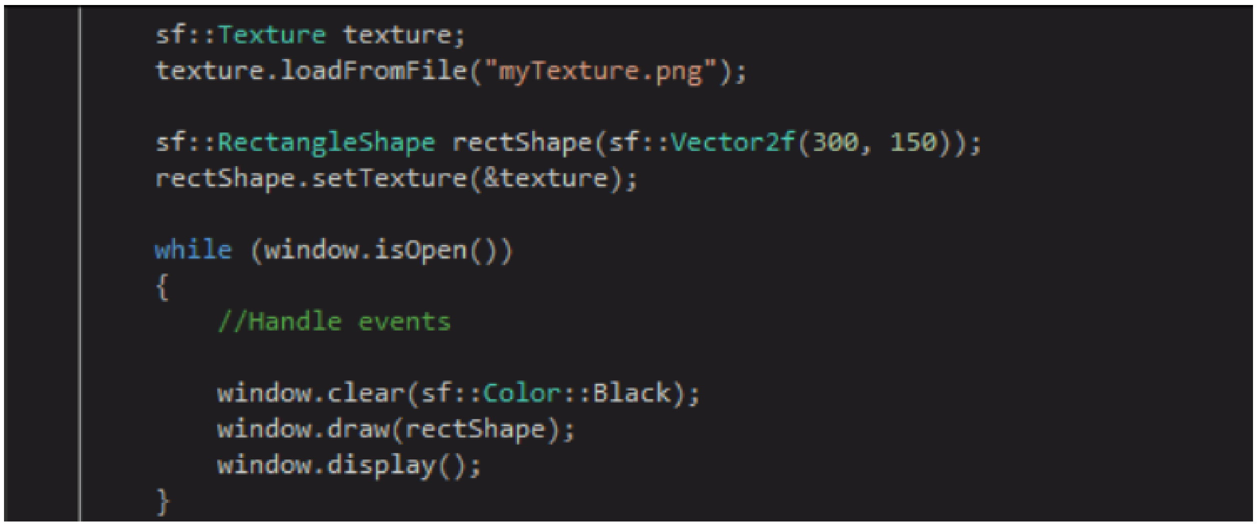
Rendering shapes with textures

Texture cannot be rendered on their own they need surface to be mapped to and then that surface can be rendered. Textures, as we mentioned in the beginning of the chapter, are just a collection of pixels, which cannot be rendered directly on the screen without having some sort of reference (such as a position, rotation, and so on).

However, in SFML, there are renderable classes which can use a texture for their surface. In fact, we used one of those classes in the previous chapter—the shape.

Apart from a fill color and an outline color, every Shape object can have a texture as well. We can apply a texture to a shape by calling Shape::setTexture() and passing a pointer to a texture. The last thing we need to do is render the shape in a window.

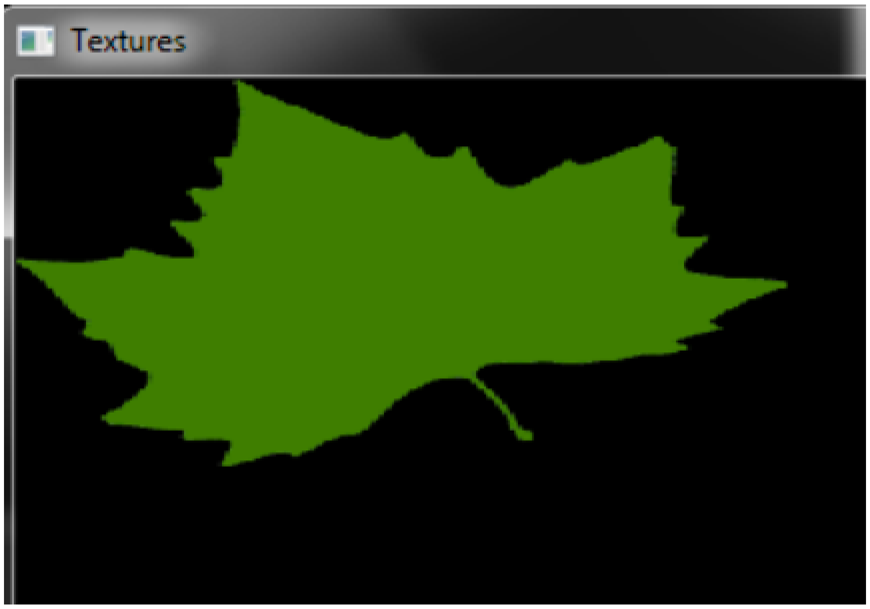


```
sf::Texture texture;
texture.loadFromFile("myTexture.png");

sf::RectangleShape rectShape(sf::Vector2f(300, 150));
rectShape.setTexture(&texture);

while (window.isOpen())
{
    //Handle events

    window.clear(sf::Color::Black);
    window.draw(rectShape);
    window.display();
}
```

The first important thing that stands out is that Shape::setTexture() takes a pointer rather than a reference. That's why we pass the address of the texture with &texture. The shape then stores that pointer locally and uses it when it needs to be rendered. This means that the address, which we pass to the function, has to hold a valid texture throughout the lifetime of the shape. Moving the texture in memory or destroying it will lead to a dangling pointer inside the Shape object, resulting
in an undefined behavior. That is why we always need to make sure that a texture does not get out of the scope of the object which uses it.

Behaviour of texture on shapes

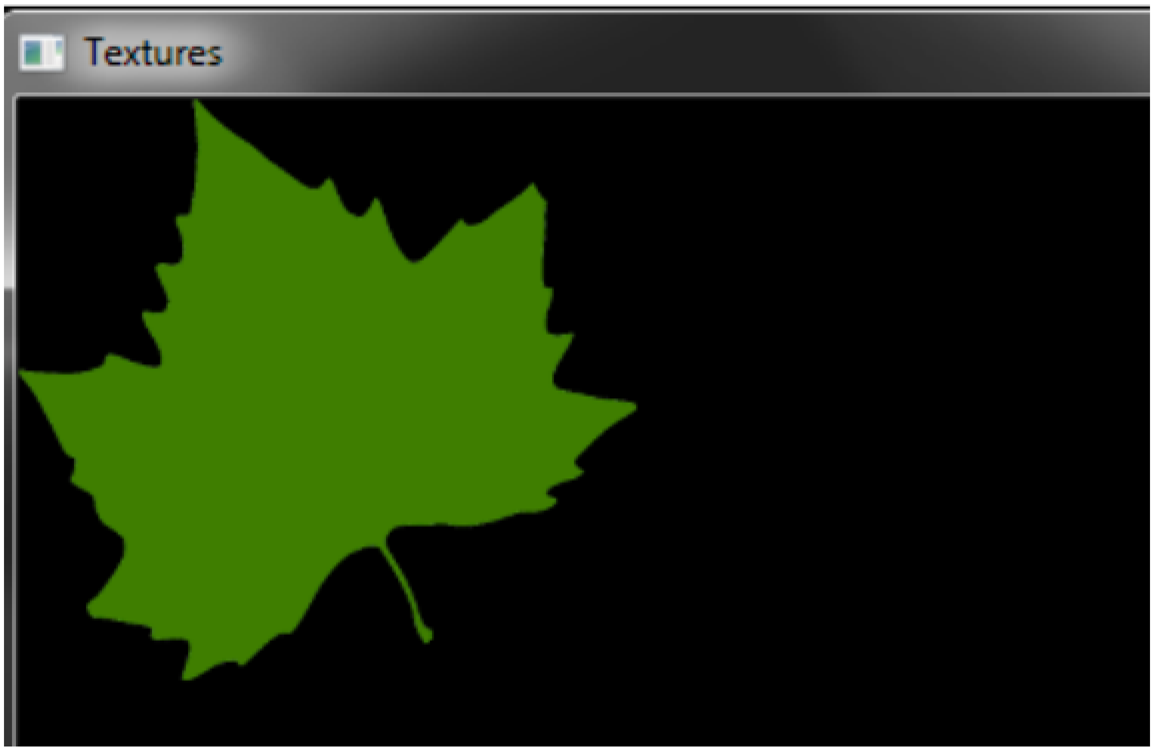
When we place a texture on RectangleShape, it tries to fit into the specified rectangle by scaling itself up or down. For our example, if the texture has a width of 200, height of 200, and the rectangle is of the size—300 as width and 150 as height, then the texture will appear stretched on the x axis and squeezed on the y axis.



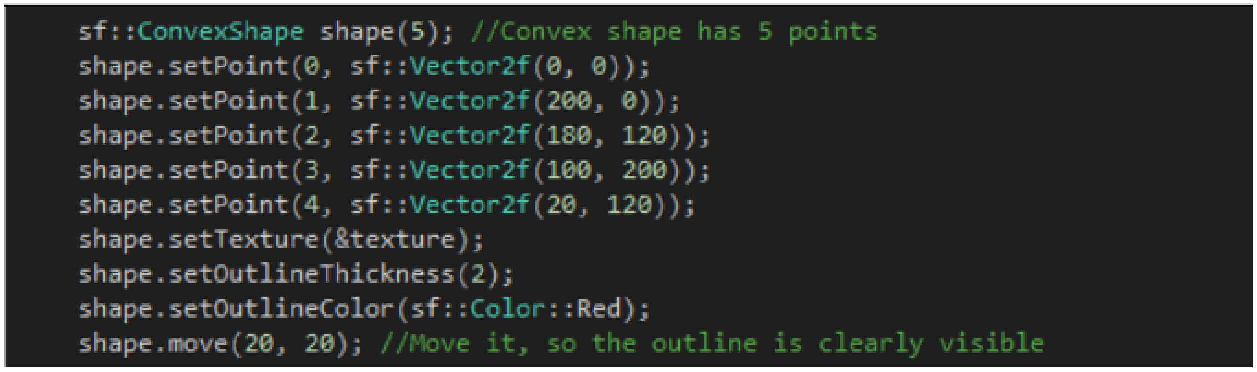
To set the shape to RectangleShape with the exact size of the texture, we can use a function from the Texture object—Texture::getSize( ).

```
    sf::Vector2u textureSize = texture.getSize();
    float rectWidth = static_cast<float>(textureSize.x);
    float rectHeight = static_cast<float>(textureSize.y);
    sf::RectangleShape rectShape(sf::Vector2f(rectWidth, rectHeight));
    rectShape.setTexture(&texture);
```

Here, the result is undistorted :

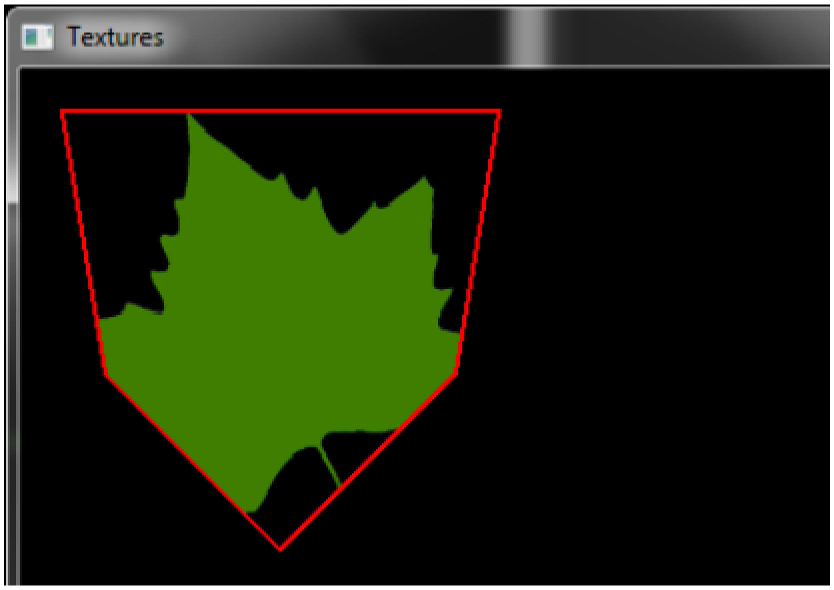


Textures can also be mapped to CircleShape and ConvexShape objects. Here is an example of how we can limit the amount of texture that is shown by using ConvexShape:



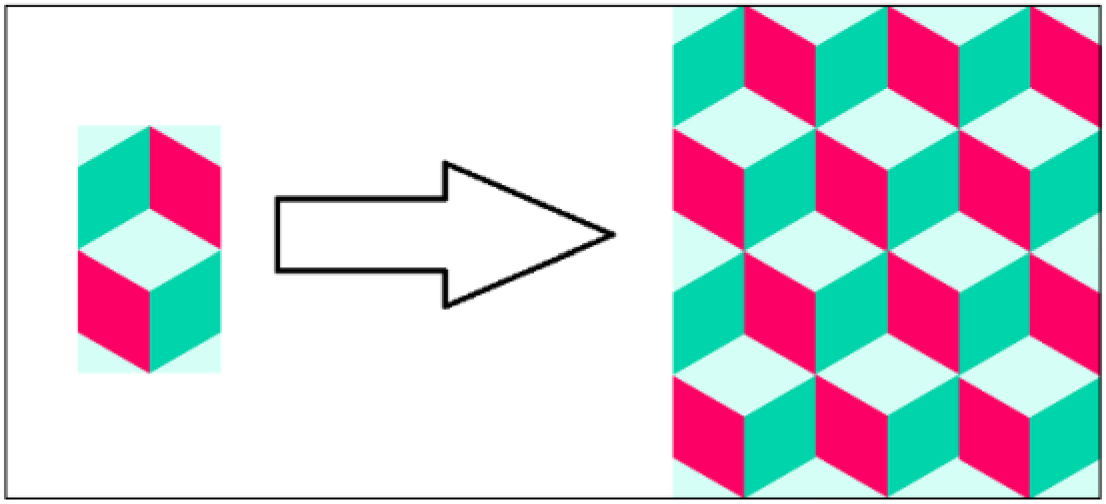
```
sf::ConvexShape shape(5); //Convex shape has 5 points
shape.setPoint(0, sf::Vector2f(0, 0));
shape.setPoint(1, sf::Vector2f(200, 0));
shape.setPoint(2, sf::Vector2f(180, 120));
shape.setPoint(3, sf::Vector2f(100, 200));
shape.setPoint(4, sf::Vector2f(20, 120));
shape.setTexture(&texture);
shape.setOutlineThickness(2);
shape.setOutlineColor(sf::Color::Red);
shape.move(20, 20); //Move it, so the outline is clearly visible
```

We will create a simple polygon with five vertices and assign it a texture. For clarity, we will also show the outline and move it a bit from the edges of the window; here is the result.



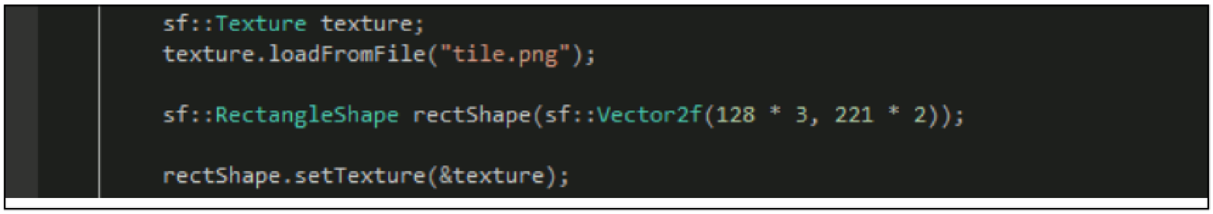
Repeat texture multiple times on a surface

Textures can also be repeated multiple times on a surface. Let's say that we want to create the following surface from this tile.



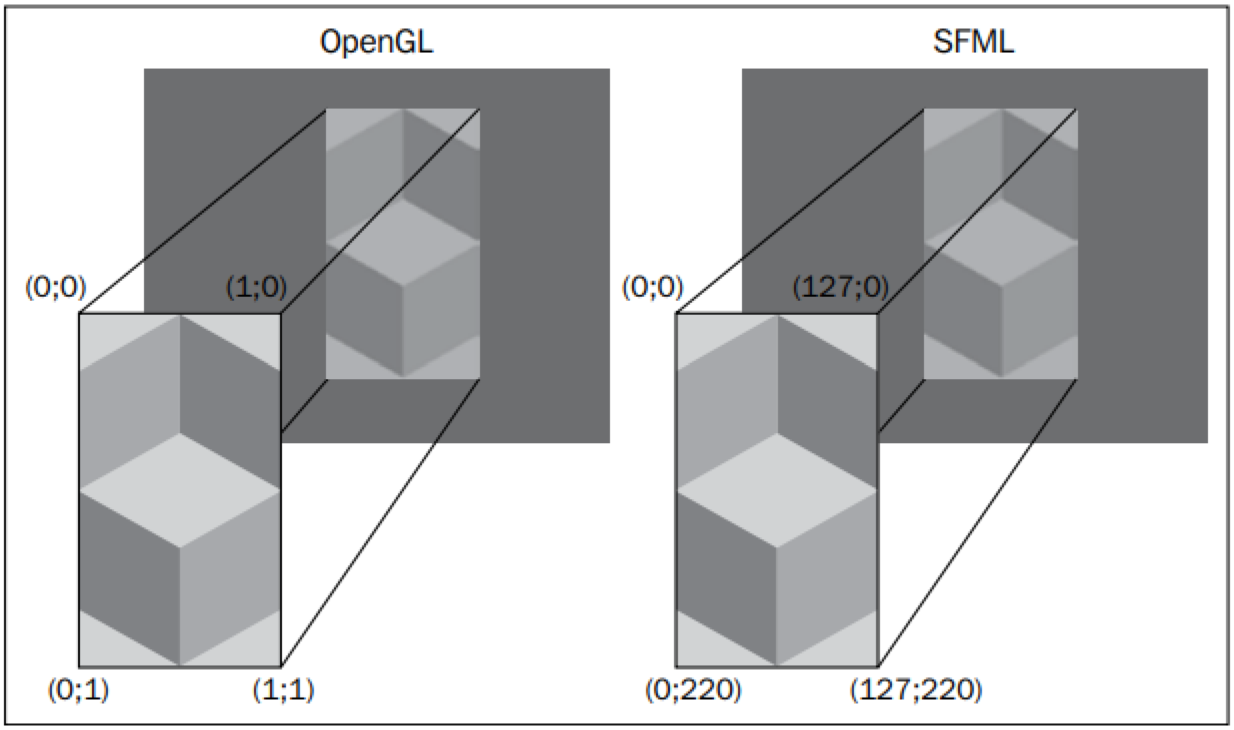
Since the tile on the left is completely seamless in all directions, we can place many of them side by side and create a bigger texture on the surface. One way of achieving this is by creating the larger image in an image editor, such as Microsoft Paint, GIMP, or Photoshop. However, that image will require more memory, which we do not necessarily want to give up. There is an alternative way where we load only the tile in a GPU memory and use it as a repeated texture over a given surface.

Our tile has dimensions 128;221, and we are replicating the tile three times on the x axis and two times on the y axis; meaning that we end up with a surface the size of 384;442. For such a shape, it is only logical to use the RectangleShape class. Here is our setup:

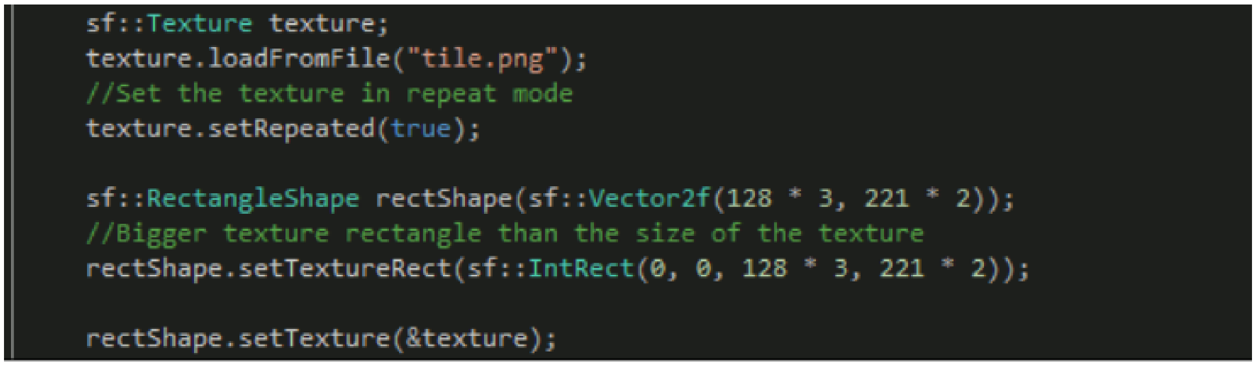


```
sf::Texture texture;
texture.loadFromFile("tile.png");

sf::RectangleShape rectShape(sf::Vector2f(128 * 3, 221 * 2));

rectShape.setTexture(&texture);
```

If we try to render the shape in its current shape, the result does not seem promising—the texture is just stretched on the whole surface of the rectangle. We need to configure the texture a bit more for it to work as we like. First of all, there is a function inside the Texture class—Texture::setRepeated(), which takes bool and marks the texture as repeatable if it is called with true. However, this is not enough; there is one more step. When we map textures to surfaces, we typically have to specify texture coordinates for each vertex of the surface. In SFML, this is done automatically for the Shape class. If we were using the OpenGL API to render a square with a texture on top of it, we would have to specify the texture coordinates in a normalized format (0…1; 0…1). SFML does not use the normalized approach; rather it uses pixel space coordinates ([0… width -1], [0… height -1]). Here is a diagram to demonstrate how texture coordinates are mapped to a surface, which is then rendered on a screen (the gray rectangle):

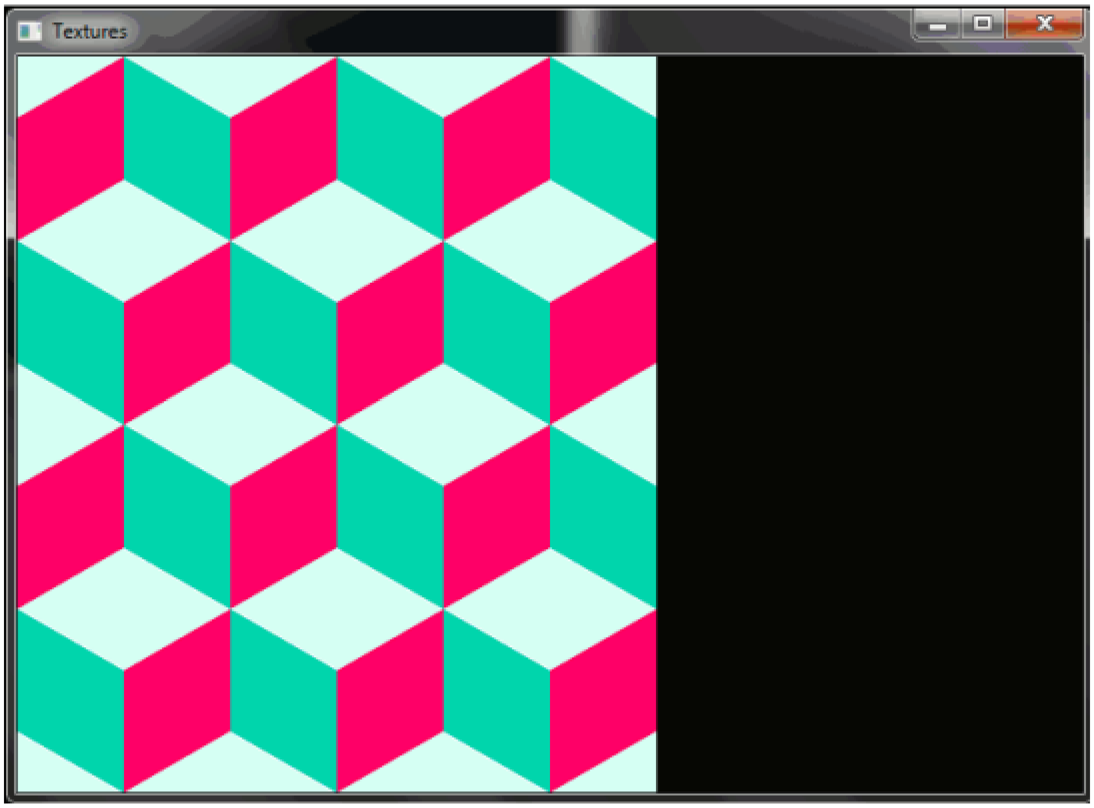


We saw what happens when we make the surface (or the shape) bigger—it just stretches the texture. We need to change the texture coordinates to repeat the texture multiple times on that surface. This is done by making the texture rectangle (all four texture coordinates in one structure) larger than the texture itself. With that in mind, let's see how our code setup will change:

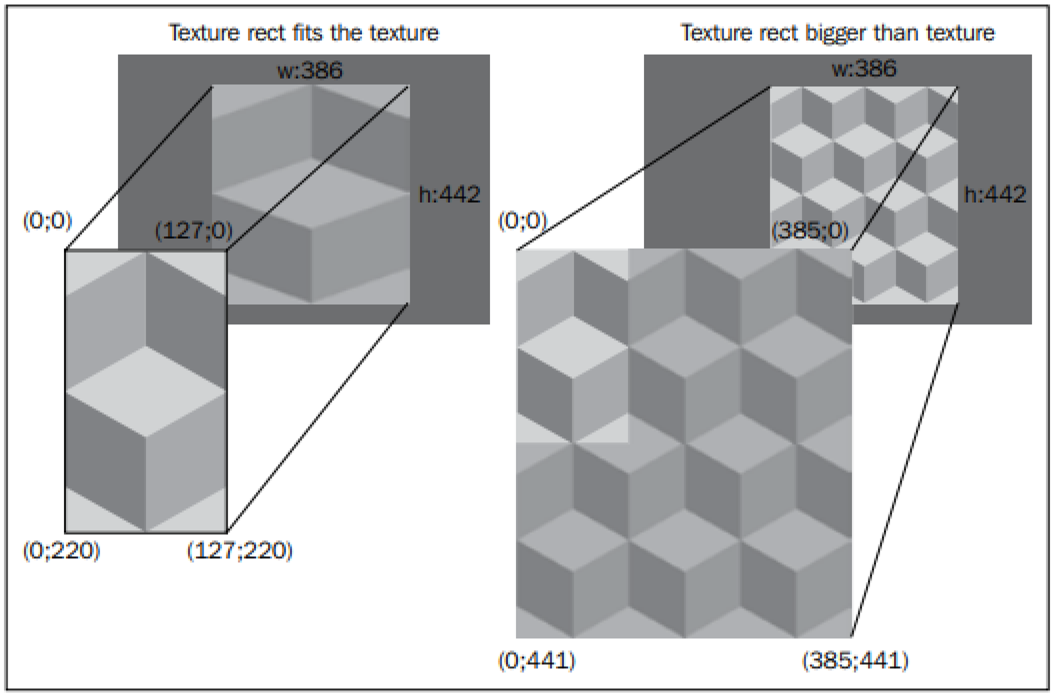


```
sf::Texture texture;
texture.loadFromFile("tile.png");
//Set the texture in repeat mode
texture.setRepeated(true);

sf::RectangleShape rectShape(sf::Vector2f(128 * 3, 221 * 2));
//Bigger texture rectangle than the size of the texture
rectShape.setTextureRect(sf::IntRect(0, 0, 128 * 3, 221 * 2));

rectShape.setTexture(&texture);
```

The result is exactly what we expected :



To clarify texture coordinates a bit further, the following diagram demonstrates how the default texture rectangle (that fits the texture perfectly) is mapped to a bigger surface, and what is the result when the texture rectangle is larger than the texture:


Texture rect fits the texture
w:386
h:442
(0;0)
(127;0)
(0;220)
(127;220)
Texture rect bigger than texture
w:386
h:442
(0;0)
(385;0)
(0;441)
(385;441)