Lição 3 – Viewport (Janela de Visualização)

Então, nós sabemos como definir o nosso sistema de coordenadas usando a matriz de projeção, mas e se apenas quisermos renderizar parte da tela? Aqui é onde a janela de visualização entra.

A *viewport* define a área retangular da tela que queremos renderizar. Aqui vamos mexer com ele para fazer coisas como a renderização em tela dividida.

1. Iniciando

Nesta lição, os códigos de *run( )* e *update( )* são iguais. A classe *Main*, contudo, possui campos diferentes dos da última lição.

```
40      public int gViewportMode = 0; //VIEW_MODE_FULL
41
42      /*
43          VIEWPORT_MODE_FULL = 0
44          VIEWPORT_MODE_HALF_CENTER = 1
45          VIEWPORT_MODE_HALF_TOP = 2
46          VIEWPORT_MODE_QUAD = 3
47          VIEWPORT_MODE_RADAR = 4
48      */
```

Podemos ver na imagem que existem algumas formas do *VIEWPORT* funcionar, e elas estão enumeradas no comentário abaixo da linha de código. Logo, sempre que quisermos alterar a forma como será renderizado o quadrado, antes devemos alterar o *viewport.*

1.1. O método *initGL( )*

```
87      public boolean initGL() {
88          //Mudar a visão
89          glViewport(0, 0, WIDTH, HEIGHT);
90
91          //Inicializar a projeção da matriz
92          glMatrixMode(GL_PROJECTION);
93          glLoadIdentity();
94          glOrtho(0.0, WIDTH, HEIGHT, 0.0, 1.0, -1.0);
95
96          //Inicializar o ModelView da matriz
97          glMatrixMode(GL_MODELVIEW);
98          glLoadIdentity();
99
100         //Inicializar a cor
101         glClearColor(0.f, 0.f, 0.f, 1.f);
102
103         //Checagem de erros
104         int erro = glGetError();
105         if(erro != GL_NO_ERROR)
106         {
107             System.out.println("Erro de inicialização do OpenGL");
108             return false;
109         }
110         return true;
111     }
```

Nosso método initGL ( ) é praticamente o mesmo de antes, mas agora ele tem uma chamada para glViewport( ) para inicializar o viewport. glViewport( ) define que parte da tela queremos renderizar definindo a coordenada x, a coordenada y, a largura e a altura da área de renderização. Como você pode ver no campo na classe *Main*, estamos apenas dizendo para renderizar a tela inteira.

1.2. O método *render( )*

```
124     public void render() throws LWJGLException {
125         //Clear color buffer
126         glClear(GL_COLOR_BUFFER_BIT);
127 
128         //Reset ModelView Matrix
129         glLoadIdentity();
130 
131         //Move to center of the display
132         glTranslatef(WIDTH / 2.f, HEIGHT / 2.f, 0.f);
133 
```

No topo da nossa função de renderização, limpamos a tela e redefinimos a matriz *modelview* como de costume. Depois disso, transladamos para o centro da tela. Para o restante de nossa renderização, note como não fazemos nenhuma outra transformação para as matrizes de projeção ou *modelview*.

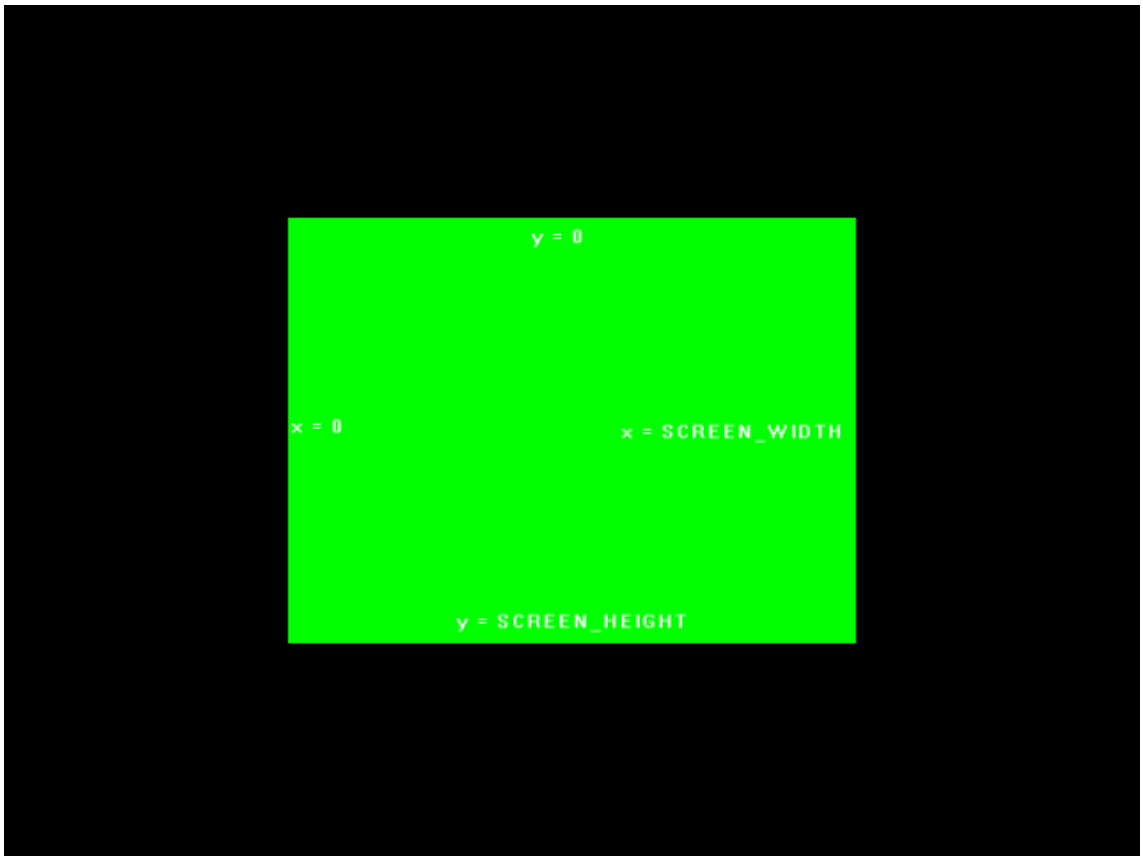
```
134         //Full View
            if(gViewportMode == 0) //VIEWPORT_MODE_FULL
136         {
137             //Preencher a tela
138             glViewport(0, 0, WIDTH, HEIGHT);
139 
140             //Quadrado vermelho
141             glBegin(GL_QUADS);
142                 glColor3f(1.f, 0.f, 0.f);
143                 glVertex2f(-WIDTH / 2.f, -HEIGHT / 2.f);
144                 glVertex2f(WIDTH / 2.f, -HEIGHT / 2.f);
145                 glVertex2f(WIDTH / 2.f, HEIGHT / 2.f);
146                 glVertex2f(-WIDTH / 2.f, HEIGHT / 2.f);
147             glEnd();
148         }
```

Em "VIEWPORT_MODE_FULL", definimos a janela de visualização como a tela inteira e renderizamos um quad em tela cheia.

Pode parecer redundante definir a viewport novamente, mas nesta demonstração vamos estar mudando a viewport dependendo de "gViewportMode".

```
150             //Visão do centro da tela
151             else if(gViewportMode == 1) //VIEWPORT_MODE_HALF_CENTER
152             {
153                 //Centro da visão
154                 glViewport(WIDTH / 4, HEIGHT / 4, WIDTH / 2, HEIGHT / 2);
155
156                 //Quadrado verde
157                 glBegin(GL_QUADS);
158                     glColor3f(0.f, 1.f, 0.f);
159                     glVertex2f(-WIDTH / 2.f, -HEIGHT / 2.f);
160                     glVertex2f(WIDTH / 2.f, -HEIGHT / 2.f);
161                     glVertex2f(WIDTH / 2.f, HEIGHT / 2.f);
162                     glVertex2f(-WIDTH / 2.f, HEIGHT / 2.f);
163                 glEnd();
164             }
165
```

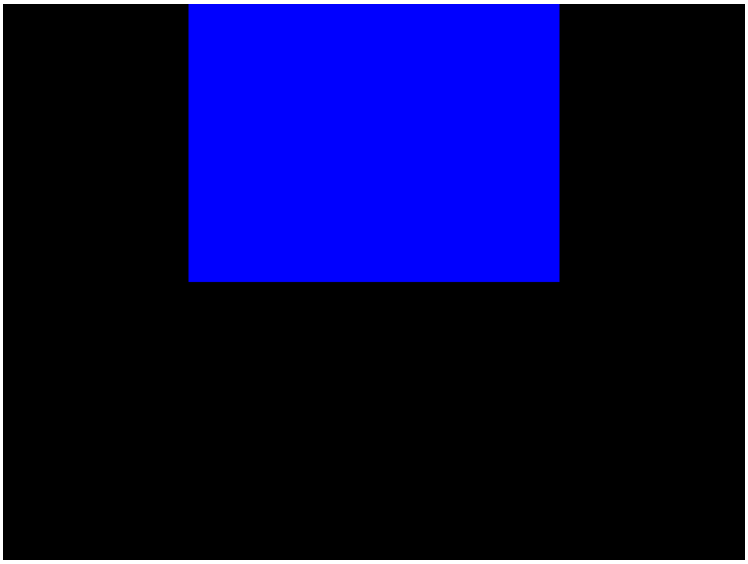
Aqui nós renderizamos o mesmo quad 640x480 em uma viewport que é metade da largura / altura da tela no meio de nossa área de renderização. Isso resulta em:



Assim, as coordenadas de renderização ainda são 640x480 mesmo se a viewport for 320x240.

```
166             //Visão centro-topo
167             else if(gViewportMode == 2) //VIEWPORT_MODE_HALF_TOP
168             {
169                 //Visão do topo
170                 glViewport(WIDTH / 4, HEIGHT / 2, WIDTH / 2, HEIGHT / 2);
171 
172                 //Quadrado azul
173                 glBegin(GL_QUADS);
174                     glColor3f(0.f, 0.f, 1.f);
175                     glVertex2f(-WIDTH / 2.f, -HEIGHT / 2.f);
176                     glVertex2f(WIDTH / 2.f, -HEIGHT / 2.f);
177                     glVertex2f(WIDTH / 2.f, HEIGHT / 2.f);
178                     glVertex2f(-WIDTH / 2.f, HEIGHT / 2.f);
179                 glEnd();
180             }
181 
```

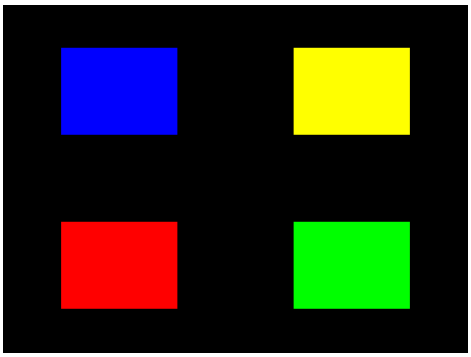
Aqui está um quadrado de 640x480, renderizado em uma viewport de 320x240, renderizado na posição da janela de visualização x = 80, y = 240. O resultado é:



Uma coisa importante a observar é que enquanto nossas coordenadas de projeção têm Y + como baixo e Y- como acima, as coordenadas de *viewport* não são garantidas. No LWJGL, Y + está para cima e Y- está para baixo para as coordenadas da *viewport*. Em alguns sistemas de janelas, Y + está para baixo e Y- é para cima para coordenadas de *viewport*. Portanto, ao trabalhar com diferentes sistemas de janelas, não assuma que as coordenadas da *viewport* são as mesmas.

```
182         //Visão dos quatro quadrados
183         else if(gViewportMode == 3) //VIEWPORT_MODE_QUAD
184         {
185             //Embaixo esquerda quadrado vermelho
186             glViewport(0, 0, WIDTH / 2, HEIGHT / 2);
187             glBegin(GL_QUADS);
188                 glColor3f(1.f, 0.f, 0.f);
189                 glVertex2f(-WIDTH / 4.f, -HEIGHT / 4.f);
190                 glVertex2f(WIDTH / 4.f, -HEIGHT / 4.f);
191                 glVertex2f(WIDTH / 4.f, HEIGHT / 4.f);
192                 glVertex2f(-WIDTH / 4.f, HEIGHT / 4.f);
193             glEnd();
194 
195             //Embaixo direita quadrado verde
196             glViewport(WIDTH / 2, 0, WIDTH / 2, HEIGHT / 2);
197             glBegin(GL_QUADS);
198                 glColor3f(0.f, 1.f, 0.f);
199                 glVertex2f(-WIDTH / 4.f, -HEIGHT / 4.f);
200                 glVertex2f(WIDTH / 4.f, -HEIGHT / 4.f);
201                 glVertex2f(WIDTH / 4.f, HEIGHT / 4.f);
202                 glVertex2f(-WIDTH / 4.f, HEIGHT / 4.f);
203             glEnd();
204 
205             //Cima esquerda quadrado azul
206             glViewport(0, HEIGHT / 2, WIDTH / 2, HEIGHT / 2);
207             glBegin(GL_QUADS);
208                 glColor3f(0.f, 0.f, 1.f);
209                 glVertex2f(-WIDTH / 4.f, -HEIGHT / 4.f);
210                 glVertex2f(WIDTH / 4.f, -HEIGHT / 4.f);
211                 glVertex2f(WIDTH / 4.f, HEIGHT / 4.f);
212                 glVertex2f(-WIDTH / 4.f, HEIGHT / 4.f);
213             glEnd();
214 
215             //Cima direita quadrado amarelo
216             glViewport(WIDTH / 2, HEIGHT / 2, WIDTH / 2, HEIGHT / 2);
217             glBegin(GL_QUADS);
218                 glColor3f(1.f, 1.f, 0.f);
219                 glVertex2f(-WIDTH / 4.f, -HEIGHT / 4.f);
220                 glVertex2f(WIDTH / 4.f, -HEIGHT / 4.f);
221                 glVertex2f(WIDTH / 4.f, HEIGHT / 4.f);
222                 glVertex2f(-WIDTH / 4.f, HEIGHT / 4.f);
223             glEnd();
224         }
```

Aqui temos várias *viewports*, que é útil para jogos de tela dividida e simulações. O mesmo quad é renderizado 4 vezes, apenas com cores diferentes e locais de *viewport*.



```
226             //Visão com radar
227             else if(gViewportMode == 4) //VIEWPORT_MODE_RADAR
228             {
229                 //Tamanho do quadrado grande
230                 glViewport(0, 0, WIDTH, HEIGHT);
231                 glBegin(GL_QUADS);
232                     glColor3f(1.f, 1.f, 1.f);
233                     glVertex2f(-WIDTH / 8.f, -HEIGHT / 8.f);
234                     glVertex2f(WIDTH / 8.f, -HEIGHT / 8.f);
235                     glVertex2f(WIDTH / 8.f, HEIGHT / 8.f);
236                     glVertex2f(-WIDTH / 8.f, HEIGHT / 8.f);
237                     glColor3f(0.f, 0.f, 0.f);
238                     glVertex2f(-WIDTH / 16.f, -HEIGHT / 16.f);
239                     glVertex2f(WIDTH / 16.f, -HEIGHT / 16.f);
240                     glVertex2f(WIDTH / 16.f, HEIGHT / 16.f);
241                     glVertex2f(-WIDTH / 16.f, HEIGHT / 16.f);
242                 glEnd();
243 
244                 //Quadrado radar
245                 glViewport(WIDTH / 2, HEIGHT / 2, WIDTH / 2, HEIGHT / 2);
246                 glBegin(GL_QUADS);
247                     glColor3f(1.f, 1.f, 1.f);
248                     glVertex2f(-WIDTH / 8.f, -HEIGHT / 8.f);
249                     glVertex2f(WIDTH / 8.f, -HEIGHT / 8.f);
250                     glVertex2f(WIDTH / 8.f, HEIGHT / 8.f);
251                     glVertex2f(-WIDTH / 8.f, HEIGHT / 8.f);
252                     glColor3f(0.f, 0.f, 0.f);
253                     glVertex2f(-WIDTH / 16.f, -HEIGHT / 16.f);
254                     glVertex2f(WIDTH / 16.f, -HEIGHT / 16.f);
255                     glVertex2f(WIDTH / 16.f, HEIGHT / 16.f);
256                     glVertex2f(-WIDTH / 16.f, HEIGHT / 16.f);
257                 glEnd();
258             }
259 
260             //Atualizando a tela
261             Display.swapBuffers();
262         }
```

E para a última das demonstrações de *viewport*, renderizamos uma cena de tamanho completo e, em seguida, renderizamos uma versão menor no canto superior esquerdo.

Ter uma *viewport* dentro de uma *viewport* pode ser útil para coisas como renderizar um radar na tela.

Claro, no final da nossa função de renderização, atualizamos a tela.

1.3. processKeyboard

```
113     public void processKeyboard() {
114         //Se pressionar a letra Q
115         if(Keyboard.isKeyDown(Keyboard.KEY_Q))
116         {
117             //Cycle through viewport modes
118             gViewportMode++;
119             if(gViewportMode > 4)
120                 gViewportMode = 0;
121         }
122     }
```

E por fim, temos nosso método de teclado, onde a cada vez que a tecla *Q* é pressionada, modificamos qual *viewport* estamos usando. Quando se chega à ultima, retorna para primeira.

Para ver outras funções do código presentes mas não explicadas, consulte as lições anteriores.