```
 1: unit Case05_06_01_HDR;
 2:
 3: {$mode objfpc}{$H+}
 4: {$ModeSwitch unicodestrings}{$J-}
 5: {$ModeSwitch advancedrecords}
 6: {$ModeSwitch implicitfunctionspecialization}
 7: {$ModeSwitch anonymousfunctions}
 8: {$ModeSwitch functionreferences}
 9:
10: interface
11:
12: uses
13:   Classes,
14:   SysUtils,
15:   DeepStar.Utils,
16:   DeepStar.OpenGL.Utils,
17:   DeepStar.OpenGL.Texture,
18:   DeepStar.OpenGL.GLAD_GL,
19:   DeepStar.OpenGL.Shader,
20:   DeepStar.OpenGL.GLM,
21:   DeepStar.OpenGL.GLFW,
22:   DeepStar.OpenGL.Camera,
23:   DeepStar.OpenGL.Model;
24:
25: procedure Main;
26:
27: implementation
28:
29: // 每当窗口大小发生变化(由操作系统或用户调整大小)，这个回调函数就会执行
30: procedure Framebuffer_size_callback(window: PGLFWwindow; witdth, Height: integer); cdecl;
    forward;
31: // 每当鼠标滚轮滚动时，这个回调就被调用
32: procedure Scroll_callback(window: PGLFWwindow; xoffset, yoffset: double); cdecl; forward;
33: // 每当鼠标移动时，就调用这个回调
34: procedure Mouse_callback(window: PGLFWwindow; xpos, ypos: double); cdecl; forward;
35: // 处理所有输入:查询GLFW是否按下/释放了相关的键，并做出相应的反应
36: procedure ProcessInput(window: PGLFWwindow); forward;
37: // glfw & glad  初始化
38: function InitWindows: PGLFWwindow; forward;
39: // 加载贴图
40: function LoadTexture(fileName: string; gammaCorrection: boolean;
41:   inverse: boolean = true): cardinal; forward;
42:
43: procedure RenderQuad; forward;
44: procedure RenderCube; forward;
45:
46: const
47:   SCR_WIDTH = 800;
48:   SCR_HEIGHT = 600;
49:
50: var
51:   camera: TCamera;
52:
53:   deltaTime: float = 0.0;  // time between current frame and last frame
54:   lastFrame: float = 0.0;
55:
56:   firstMouse: boolean = true;
57:
58:   //偏航被初始化为-90.0度，因为0.0的偏航导致一个指向右的方向矢量，所以我们最初
59:   //向左旋转一点。
60:   lastX: float = SCR_WIDTH / 2.0;
61:   lastY: float = SCR_HEIGHT / 2.0;
62:
```

```
 63:   hdr: Boolean = true;
 64:   hdrKeyPressed: Boolean = false;
 65:   exposure: float = 1.0;
 66:
 67:   cubeVAO: Cardinal = 0;
 68:   cubeVBO: Cardinal = 0;
 69:
 70:   quadVAO: Cardinal = 0;
 71:   quadVBO: Cardinal = 0;
 72:
 73: procedure Main;
 74: const
 75:   dir_path = '..\Source\5.Advanced_Lighting\6.1.HDR\';
 76:   hdr_vs = dir_path + '6.hdr.vs';
 77:   hdr_fs = dir_path + '6.hdr.fs';
 78:   lighting_vs = dir_path + '6.lighting.vs';
 79:   lighting_fs = dir_path + '6.lighting.fs';
 80:   img_wood = '..\Resources\textures\wood.png';
 81: var
 82:   window: PGLFWwindow;
 83:   currentFrame: GLfloat;
 84:   projection, view, model: TMat4;
 85:   diffuseMap, normalMap, hdrFBO, colorBuffer, rboDepth, woodTexture: Cardinal;
 86:   shader, hdrShader: TShaderProgram;
 87:   shader_managed, camera_managed, hdrShader_managed: IInterface;
 88:   lightPosition_managed, lightColors_managed: IInterface;
 89:   lightPositions, lightColors: TArrayList_TVec3;
 90:   i: Integer;
 91: begin
 92:   window := InitWindows;
 93:   if window = nil then
 94:   begin
 95:     glfwTerminate;
 96:     Exit;
 97:   end;
 98:
 99:   //============================================================================
100:
101:   // configure global opengl state
102:   glEnable(GL_DEPTH_TEST);
103:
104:   //============================================================================
105:
106:   shader_managed := IInterface(TShaderProgram.Create);
107:   shader := shader_managed as TShaderProgram;
108:   shader.LoadShaderFile(lighting_vs, lighting_fs);
109:
110:   hdrShader_managed := IInterface(TShaderProgram.Create);
111:   hdrShader := hdrShader_managed as TShaderProgram;
112:   hdrShader.LoadShaderFile(hdr_vs, hdr_fs);
113:
114:   camera_managed := IInterface(TCamera.Create(TGLM.Vec3(0, 0, 5)));
115:   camera := camera_managed as TCamera;
116:
117:   //============================================================================
118:
119:   // load textures
120:   woodTexture := LoadTexture(img_wood, true);
121:
122:   //============================================================================
123:
124:   // 配置浮点帧缓冲区
125:   hdrFBO := Cardinal(0);
```

```
126:   glGenFramebuffers(1, @hdrFBO);
127:
128:   // 创建浮点颜色缓冲
129:   colorBuffer := Cardinal(0);
130:   glGenTextures(1, @colorBuffer);
131:   glBindTexture(GL_TEXTURE_2D, colorBuffer);
132:   glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA16F, SCR_WIDTH, SCR_HEIGHT, 0, GL_RGBA, GL_FLOAT, nil
      );
133:   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
134:   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
135:
136:   // 创建深度缓冲(renderbuffer)
137:   rboDepth := Cardinal(0);
138:   glGenRenderbuffers(1, @rboDepth);
139:   glBindRenderbuffer(GL_RENDERBUFFER, rboDepth);
140:   glRenderbufferStorage(GL_RENDERBUFFER, GL_DEPTH_COMPONENT, SCR_WIDTH, SCR_HEIGHT);
141:
142:   // 附加缓冲
143:   glBindFramebuffer(GL_FRAMEBUFFER, hdrFBO);
144:   glFramebufferTexture2D(GL_FRAMEBUFFER, GL_COLOR_ATTACHMENT0, GL_TEXTURE_2D, colorBuffer, 0)
      ;
145:   glFramebufferRenderbuffer(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, GL_RENDERBUFFER, rboDepth);
146:   if glCheckFramebufferStatus(GL_FRAMEBUFFER) <> GL_FRAMEBUFFER_COMPLETE then
147:     WriteLn('Framebuffer not complete!');
148:   glBindFramebuffer(GL_FRAMEBUFFER, 0);
149:
150:   //============================================================================
151:   // lighting info
152:
153:   // position
154:   lightPosition_managed := IInterface(TArrayList_TVec3.Create);
155:   lightPositions := lightPosition_managed as TArrayList_TVec3;
156:   lightPositions.AddLast(TGLM.Vec3( 0.0,  0.0, 49.5));
157:   lightPositions.AddLast(TGLM.Vec3(-1.4, -1.9,  9.0));
158:   lightPositions.AddLast(TGLM.Vec3( 0.0, -1.8,  4.0));
159:   lightPositions.AddLast(TGLM.Vec3( 0.8, -1.7,  6.0));
160:
161:   // colors
162:   lightColors_managed := IInterface(TArrayList_TVec3.Create);
163:   lightColors := lightColors_managed as TArrayList_TVec3;
164:   lightColors.AddFirst(TGLM.Vec3(200.0, 200.0, 200.0));
165:   lightColors.AddFirst(TGLM.Vec3(  0.1,   0.0,   0.0));
166:   lightColors.AddFirst(TGLM.Vec3(  0.0,   0.0,   0.2));
167:   lightColors.AddFirst(TGLM.Vec3(  0.0,   0.1,   0.0));
168:
169:   //============================================================================
170:
171:   // shader configuration
172:   shader.UseProgram;
173:   shader.SetUniformInt('diffuseTexture', 0);
174:
175:   hdrShader.UseProgram;
176:   hdrShader.SetUniformInt('hdrBuffer', 0);
177:
178:   //============================================================================
179:
180:   // 渲染循环
181:   while not glfwWindowShouldClose(window).ToBoolean do
182:   begin
183:     // 每帧时时逻辑
184:     currentFrame := GLfloat(glfwGetTime);
185:     deltaTime := currentFrame - lastFrame;
186:     lastFrame := currentFrame;
```

```
187:
188:     // 输入
189:     ProcessInput(window);
190:
191:     // render
192:     glClearColor(0.1, 0.1, 0.1, 1.0);
193:     glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
194:
195:     // 1. 渲染场景到浮点帧缓冲区
196:     glBindFramebuffer(GL_FRAMEBUFFER, hdrFBO);
197:       glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
198:       projection := TGLM.Perspective(TGLM.Radians(camera.Zoom), SCR_WIDTH / SCR_HEIGHT,
199:         0.1, 100.0);
200:       view := camera.GetViewMatrix;
201:       shader.UseProgram;
202:       shader.SetUniformMatrix4fv('projection', projection);
203:       shader.SetUniformMatrix4fv('view', view);
204:       glActiveTexture(GL_TEXTURE0);
205:       glBindTexture(GL_TEXTURE_2D, woodTexture);
206:
207:       // 设置 lighting uniforms
208:       for i := 0 to lightPositions.Count - 1 do
209:       begin
210:         shader.SetUniformVec3('lights[' + i.ToString + '].Position', lightPositions[i]);
211:         shader.SetUniformVec3('lights[' + i.ToString + '].Color', lightColors[i]);
212:       end;
213:       shader.SetUniformVec3('viewPos', camera.Position);
214:
215:       //渲染隧道
216:       model := TGLM.Mat4(1.0);
217:       model := TGLM.Translate(model, TGLM.Vec3(0.0, 0.0, 25));
218:       model := TGLM.Scale(model, TGLM.Vec3(2.5, 2.5, 27.5));
219:       shader.SetUniformMatrix4fv('model', model);
220:       shader.SetUniformInt('inverse_normals', true.ToInteger);
221:       RenderCube;
222:     glBindFramebuffer(GL_FRAMEBUFFER, 0);
223:
224:     // 2. 现在将浮点颜色缓冲区渲染为2D四边形,
225:     // 并将色调映射HDR颜色渲染为默认framebuffer的(固定)颜色范围
226:     glClear(GL_COLOR_BUFFER_BIT or GL_DEPTH_BUFFER_BIT);
227:     hdrShader.UseProgram;
228:     glActiveTexture(GL_TEXTURE0);
229:     glBindTexture(GL_TEXTURE_2D, colorBuffer);
230:     hdrShader.SetUniformInt('hdr', hdr.ToInteger);
231:     hdrShader.SetUniformFloat('exposure', exposure);
232:     renderQuad;
233:
234:     WriteLn('HDR: ', IfThen(hdr, 'on', 'off'), '| exposure: ', exposure.ToString);
235:
236:     //═══════════════════════════════════════════════════════════════════
237:
238:     // 交换缓冲区和轮询IO事件(键按/释放，鼠标移动等)。
239:     glfwSwapBuffers(window);
240:     glfwPollEvents;
241:   end;
242:
243:   glDeleteVertexArrays(1, @quadVAO);
244:   glDeleteBuffers(1, @quadVBO);
245:
246:   glDeleteTextures(1, @diffuseMap);
247:   glDeleteTextures(1, @normalMap);
248:
249:   // 释放 / 删除之前的分配的所有资源
```

```
250:   glfwTerminate;
251: end;
252:
253: function InitWindows: PGLFWwindow;
254: var
255:   window: PGLFWwindow = nil;
256: begin
257:   if not glfwInit.ToBoolean then Exit(nil);
258:
259:   // 设置主要版本和次要版本
260:   glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
261:   glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
262:   glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
263:
264:   // 创建一个窗口对象
265:   window := glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT, string('LearnOpenGL'), nil, nil);
266:   if window = nil then
267:   begin
268:     WriteLn('Failed to create GLFW window');
269:     Exit(nil);
270:   end;
271:
272:   // 将窗口的上下文设置为当前线程的主上下文
273:   glfwMakeContextCurrent(window);
274:
275:   // 初始化GLAD
276:   if gladLoadGL(TLoadProc(@glfwGetProcAddress)) = false then
277:   begin
278:     WriteLn('Failed to initialize GLAD');
279:     Exit(nil);
280:   end;
281:
282:   // 设置窗口的维度(Dimension)
283:   glViewport(0, 0, SCR_WIDTH, SCR_HEIGHT);
284:
285:   glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);
286:
287:   // 注册一个回调函数(Callback Function)，它会在每次窗口大小被调整的时候被调用
288:   glfwSetFramebufferSizeCallback(window, @Framebuffer_size_callback);
289:   glfwSetCursorPosCallback(window, @Mouse_callback);
290:   glfwSetScrollCallback(window, @Scroll_callback);
291:
292:   Result := window;
293: end;
294:
295: function LoadTexture(fileName: string; gammaCorrection: boolean; inverse: boolean): cardinal;
296: var
297:   texture_ID: GLuint;
298:   tx: TTexture;
299:   tx_managed: IInterface;
300:   internalFormat: GLenum;
301: begin
302:   texture_ID := GLuint(0);
303:   glGenTextures(1, @texture_ID);
304:   internalFormat := GLenum(0);
305:
306:   tx_managed := IInterface(TTexture.Create);
307:   tx := tx_managed as TTexture;
308:
309:   tx.LoadFormFile(fileName, inverse);
310:
311:   if gammaCorrection then
312:     internalFormat := GL_SRGB
```

```
313:   else
314:     internalFormat := GL_RGBA;
315:
316:   glBindTexture(GL_TEXTURE_2D, texture_ID);
317:   glTexImage2D(GL_TEXTURE_2D, 0, internalFormat, tx.Width, tx.Height, 0, GL_RGBA,
318:     GL_UNSIGNED_BYTE, tx.Pixels);
319:   glGenerateMipmap(GL_TEXTURE_2D);
320:
321:   if tx.UseAlpha then
322:   begin
323:     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
324:     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
325:   end
326:   else
327:   begin
328:     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
329:     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
330:   end;
331:
332:   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
333:   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
334:
335:   Result := texture_ID;
336: end;
337:
338: procedure RenderQuad;
339: var
340:   quadVertices: TArr_GLfloat;
341: begin
342:   if quadVAO = 0 then
343:   begin
344:     quadVertices := TArr_GLfloat([
345:       // positions        // texture Coords
346:       -1.0,  1.0, 0.0,    0.0, 1.0,
347:       -1.0, -1.0, 0.0,    0.0, 0.0,
348:        1.0,  1.0, 0.0,    1.0, 1.0,
349:        1.0, -1.0, 0.0,    1.0, 0.0]);
350:
351:     // setup plane VAO
352:     glGenVertexArrays(1, @quadVAO);
353:     glGenBuffers(1, @quadVBO);
354:     glBindVertexArray(quadVAO);
355:     glBindBuffer(GL_ARRAY_BUFFER, quadVBO);
356:     glBufferData(GL_ARRAY_BUFFER, quadVertices.MemSize, @quadVertices[0], GL_STATIC_DRAW);
357:     glEnableVertexAttribArray(0);
358:     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 5 * SIZE_OF_F, Pointer(0));
359:     glEnableVertexAttribArray(1);
360:     glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 5 * SIZE_OF_F, Pointer(3 * SIZE_OF_F));
361:   end;
362:
363:   glBindVertexArray(quadVAO);
364:   glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
365:   glBindVertexArray(0);
366: end;
367:
368: procedure RenderCube;
369: var
370:   vertices: TArr_GLfloat;
371: begin
372:   if cubeVAO = 0 then
373:   begin
374:     vertices := TArr_GLfloat([
375:       // back face
```

```
376:       -1.0, -1.0, -1.0,  0.0,  0.0, -1.0, 0.0, 0.0, // bottom-left
377:        1.0,  1.0, -1.0,  0.0,  0.0, -1.0, 1.0, 1.0, // top-right
378:        1.0, -1.0, -1.0,  0.0,  0.0, -1.0, 1.0, 0.0, // bottom-right
379:        1.0,  1.0, -1.0,  0.0,  0.0, -1.0, 1.0, 1.0, // top-right
380:       -1.0, -1.0, -1.0,  0.0,  0.0, -1.0, 0.0, 0.0, // bottom-left
381:       -1.0,  1.0, -1.0,  0.0,  0.0, -1.0, 0.0, 1.0, // top-left
382:       // front face
383:       -1.0, -1.0,  1.0,  0.0,  0.0,  1.0, 0.0, 0.0, // bottom-left
384:        1.0, -1.0,  1.0,  0.0,  0.0,  1.0, 1.0, 0.0, // bottom-right
385:        1.0,  1.0,  1.0,  0.0,  0.0,  1.0, 1.0, 1.0, // top-right
386:        1.0,  1.0,  1.0,  0.0,  0.0,  1.0, 1.0, 1.0, // top-right
387:       -1.0,  1.0,  1.0,  0.0,  0.0,  1.0, 0.0, 1.0, // top-left
388:       -1.0, -1.0,  1.0,  0.0,  0.0,  1.0, 0.0, 0.0, // bottom-left
389:       // left face
390:       -1.0,  1.0,  1.0, -1.0,  0.0,  0.0, 1.0, 0.0, // top-right
391:       -1.0,  1.0, -1.0, -1.0,  0.0,  0.0, 1.0, 1.0, // top-left
392:       -1.0, -1.0, -1.0, -1.0,  0.0,  0.0, 0.0, 1.0, // bottom-left
393:       -1.0, -1.0, -1.0, -1.0,  0.0,  0.0, 0.0, 1.0, // bottom-left
394:       -1.0, -1.0,  1.0, -1.0,  0.0,  0.0, 0.0, 0.0, // bottom-right
395:       -1.0,  1.0,  1.0, -1.0,  0.0,  0.0, 1.0, 0.0, // top-right
396:       // right face
397:        1.0,  1.0,  1.0,  1.0,  0.0,  0.0, 1.0, 0.0, // top-left
398:        1.0, -1.0, -1.0,  1.0,  0.0,  0.0, 0.0, 1.0, // bottom-right
399:        1.0,  1.0, -1.0,  1.0,  0.0,  0.0, 1.0, 1.0, // top-right
400:        1.0, -1.0, -1.0,  1.0,  0.0,  0.0, 0.0, 1.0, // bottom-right
401:        1.0,  1.0,  1.0,  1.0,  0.0,  0.0, 1.0, 0.0, // top-left
402:        1.0, -1.0,  1.0,  1.0,  0.0,  0.0, 0.0, 0.0, // bottom-left
403:       // bottom face
404:       -1.0, -1.0, -1.0,  0.0, -1.0,  0.0, 0.0, 1.0, // top-right
405:        1.0, -1.0, -1.0,  0.0, -1.0,  0.0, 1.0, 1.0, // top-left
406:        1.0, -1.0,  1.0,  0.0, -1.0,  0.0, 1.0, 0.0, // bottom-left
407:        1.0, -1.0,  1.0,  0.0, -1.0,  0.0, 1.0, 0.0, // bottom-left
408:       -1.0, -1.0,  1.0,  0.0, -1.0,  0.0, 0.0, 0.0, // bottom-right
409:       -1.0, -1.0, -1.0,  0.0, -1.0,  0.0, 0.0, 1.0, // top-right
410:       // top face
411:       -1.0,  1.0, -1.0,  0.0,  1.0,  0.0, 0.0, 1.0, // top-left
412:        1.0,  1.0 , 1.0,  0.0,  1.0,  0.0, 1.0, 0.0, // bottom-right
413:        1.0,  1.0, -1.0,  0.0,  1.0,  0.0, 1.0, 1.0, // top-right
414:        1.0,  1.0,  1.0,  0.0,  1.0,  0.0, 1.0, 0.0, // bottom-right
415:       -1.0,  1.0, -1.0,  0.0,  1.0,  0.0, 0.0, 1.0, // top-left
416:       -1.0,  1.0,  1.0,  0.0,  1.0,  0.0, 0.0, 0.0  // bottom-left
417:       ]);
418:
419:     glGenVertexArrays(1, @cubeVAO);
420:     glGenBuffers(1, @cubeVBO);
421:
422:     // fill buffer
423:     glBindBuffer(GL_ARRAY_BUFFER, cubeVBO);
424:     glBufferData(GL_ARRAY_BUFFER, vertices.MemSize, @vertices[0], GL_STATIC_DRAW);
425:
426:     // link vertex attributes
427:     glBindVertexArray(cubeVAO);
428:     glEnableVertexAttribArray(0);
429:     glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 8 * SIZE_OF_F, Pointer(0));
430:     glEnableVertexAttribArray(1);
431:     glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 8 * SIZE_OF_F, Pointer(3 * SIZE_OF_F));
432:     glEnableVertexAttribArray(2);
433:     glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * SIZE_OF_F, Pointer(6 * SIZE_OF_F));
434:     glBindBuffer(GL_ARRAY_BUFFER, 0);
435:     glBindVertexArray(0);
436:   end;
437:
438:   // render Cube
```

```
439:   glBindVertexArray(cubeVAO);
440:   glDrawArrays(GL_TRIANGLES, 0, 36);
441:   glBindVertexArray(0);
442: end;
443:
444: procedure Framebuffer_size_callback(window: PGLFWwindow; witdth, Height: integer); cdecl;
445: begin
446:   //确保视口匹配新的窗口尺寸;注意宽度和
447:   //高度将明显大于视网膜显示器上的指定。
448:   glViewport(0, 0, witdth, Height);
449: end;
450:
451: procedure ProcessInput(window: PGLFWwindow);
452: begin
453:   if glfwGetKey(window, GLFW_KEY_ESCAPE) = GLFW_PRESS then
454:     glfwSetWindowShouldClose(window, true.ToInteger);
455:
456:   if glfwGetKey(window, GLFW_KEY_W) = GLFW_PRESS then
457:     camera.ProcessKeyboard(TCamera_Movement.FORWARD, deltaTime);
458:   if glfwGetKey(window, GLFW_KEY_S) = GLFW_PRESS then
459:     camera.ProcessKeyboard(TCamera_Movement.BACKWARD, deltaTime);
460:   if glfwGetKey(window, GLFW_KEY_A) = GLFW_PRESS then
461:     camera.ProcessKeyboard(TCamera_Movement.LEFT, deltaTime);
462:   if glfwGetKey(window, GLFW_KEY_D) = GLFW_PRESS then
463:     camera.ProcessKeyboard(TCamera_Movement.RIGHT, deltaTime);
464:
465:   if (glfwGetKey(window, GLFW_KEY_SPACE) = GLFW_PRESS) and (not hdrKeyPressed) then
466:   begin
467:     hdr := not hdr;
468:     hdrKeyPressed := true;
469:   end
470:   else if glfwGetKey(window, GLFW_KEY_SPACE) = GLFW_RELEASE then
471:   begin
472:     hdrKeyPressed := false;
473:   end;
474:
475:   if glfwGetKey(window, GLFW_KEY_Q) = GLFW_PRESS then
476:   begin
477:     if exposure > 0.0 then
478:       exposure -= 0.0001
479:     else
480:       exposure := 0.0;
481:   end
482:   else if glfwGetKey(window, GLFW_KEY_E) = GLFW_PRESS then
483:   begin
484:     if exposure < 1.0 then
485:       exposure += 0.0001
486:     else
487:       exposure := 1.0;
488:   end;
489: end;
490:
491: procedure Mouse_callback(window: PGLFWwindow; xpos, ypos: double); cdecl;
492: var
493:   xoffset, yoffset: GLfloat;
494: begin
495:   if firstMouse then
496:   begin
497:     lastX := xpos;
498:     lastY := ypos;
499:     firstMouse := false;
500:   end;
501:
```

```
502:   xoffset := GLfloat(xpos - lastX);
503:   yoffset := GLfloat(lastY - ypos);
504:   lastX := xpos;
505:   lastY := ypos;
506:
507:   camera.ProcessMouseMovement(xoffset, yoffset);
508: end;
509:
510: procedure Scroll_callback(window: PGLFWwindow; xoffset, yoffset: double); cdecl;
511: begin
512:   camera.ProcessMouseScroll(yoffset);
513: end;
514:
515: end.
```