山东大学 计算机 学院

计算机图形学 课程实验报告

|  |  |  |  |
| --- | --- | --- | --- |
| 学号：201400301279 | 姓名： 周鸿 | | 班级： 2014级菁英班 |
| 实验题目：  实验三：Raytracing | | | |
| 实验学时：4 | | 实验日期：2017.5.1 | |
| 实验目的：将光线追踪算法raytracing应用于具体的物体上。 | | | |
| 硬件环境：  华硕笔记本 | | | |
| 软件环境：  Visual studio 2015  OpenGl | | | |
| 实验步骤与内容：   1. 实验准备：   光线跟踪的基础知识：从一个照相机发射出光线，让这些光线穿过一个屏幕平面进入到场景中，跟踪光线从而寻找出其与几何体最近的相交点，并简单地使用一个点积值来计算散射度，最终完成对像素点颜色值的计算。简图如下：  图中的蓝线是反射光线，绿线是折射光线。后者比前者要难计算，但也是可以做的。它的计算主要涉及到折射因子和折射定律。红线是用来探测光源的。一般来说，如果你想计算散射光，那么若对于相交点来说光源是可见的，就对点积乘以1，否则就乘以0，将其排除出去。当然若光源是半可见就乘以0.5.每条新产生的光线都增加了它先前光线聚集的地方的颜色，最终每条光线都对最开始由primary ray穿过的像素点的颜色做出了自己的贡献。   1. 实验构思:   在这里我将在上一个实验即实验二的基础上对球体的phong光照模型进行raytracing   1. 具体实现：   增加了Phong光照模型的计算；使用阴影线secondary ray，这种光线和其他的不同：它对于产生它的光线的颜色没有贡献；相反，它们经常用来判段一个光源是否可以“看见”一个相交点。最后得到的效果为两个光线跟踪的球体，带反射，散射，和镜面光照，还有来自两个光源产生的阴影。关键代码如下：  #include <string.h>  //extern int main(int n,char \*\*o); /\* real main (well...) \*/  int HW\_cmd\_show;  HINSTANCE HW\_instance;  char HW\_class\_name[] = "RayTracing windows";  HWND HW\_wnd; /\* window \*/  HPALETTE HW\_palette; /\* the palette headers \*/  HDC HW\_mem;  HBITMAP HW\_bmp;  RECT HW\_rect;  #if defined(\_RGB\_) /\* paths to data sets \*/  #if defined(\_32BPP\_)  char path[128]="";  #endif  #endif  struct TR\_world \*w;  void HWI\_null\_function(void) {}  void (\*HW\_application\_main)(void) = HWI\_null\_function;  void (\*HW\_application\_key\_handler)(int key\_code);  LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam);  #define HW\_MAX\_ERROR\_MESSAGE 256  void HW\_error(char \*s,...)  {  char str[HW\_MAX\_ERROR\_MESSAGE];  va\_list lst;  va\_start(lst,s);  vsprintf(str,s,lst); /\* forming the message \*/  va\_end(lst);  MessageBox(NULL,str,"3D RayTracing",MB\_OK|MB\_ICONSTOP|MB\_SYSTEMMODAL);  HW\_close\_event\_loop();  exit(0); /\* above might not be enough \*/  }  void HW\_close\_event\_loop(void)  {  PostMessage(HW\_wnd,WM\_CLOSE,0,0L); /\* telling ourselves to quit \*/  }  void HW\_set\_int(int \*d,long l,int v)  {  long i;  for(i=0;i<l;i++) \*d++=v;  }  #define HW\_MAX\_CLINE\_OPTIONS 20  int WINAPI WinMain(HINSTANCE hInstance,  HINSTANCE hPrevInstance,  LPSTR lpCmdLine,  int nShowCmd )  {  WNDCLASS w;  int n;  char \*start,\*end;  char \*o[HW\_MAX\_CLINE\_OPTIONS];  HW\_cmd\_show = nShowCmd;  if ((HW\_instance = hPrevInstance) == NULL)  {  w.style = CS\_HREDRAW|CS\_VREDRAW;  w.lpfnWndProc = WndProc;  w.cbClsExtra = 0;  w.cbWndExtra = 0;  w.hInstance = hInstance;  w.hIcon = NULL;  w.hCursor = NULL;  w.hbrBackground = (HBRUSH)GetStockObject(WHITE\_BRUSH);  w.lpszMenuName = NULL;  w.lpszClassName = HW\_class\_name;  if (!RegisterClass(&w))  {  return FALSE;  }  }  n = 0;  o[n++] = "";  start = lpCmdLine;  while ((end = strchr(start, ' ')) != NULL)  {  if (n >= HW\_MAX\_CLINE\_OPTIONS)  {  HW\_error("(RayTracing windows) Way too many command line options.\n");  }  if (end != start)  {  o[n++] = start;  }  \*end = 0;  start = end + 1;  }  if (strlen(start) > 0)  {  o[n++] = start;  }  return(main(n,0));  }  LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)  {  switch(message)  {  case WM\_PAINT:  HW\_application\_main();  break;  case WM\_ERASEBKGND:  return(1L);  break;  case WM\_DESTROY:  PostQuitMessage(0);  break;  case WM\_KEYDOWN:  HW\_application\_key\_handler(wParam);  break;  default:  return(DefWindowProc(hWnd,message,wParam,lParam));  }  return(0L);  }  #if defined(\_RGB\_)  void HW\_init\_screen(char \*display\_name,  char \*screen\_name  )  #endif  {  PAINTSTRUCT ps;  int i,remap;  HW\_wnd = CreateWindow(HW\_class\_name,  screen\_name,  WS\_SYSMENU,  CW\_USEDEFAULT,  CW\_USEDEFAULT,  HW\_SCREEN\_X\_SIZE, HW\_SCREEN\_Y\_SIZE+GetSystemMetrics(SM\_CYCAPTION),  NULL,  NULL,  HW\_instance,  NULL);  HW\_mem=CreateCompatibleDC(BeginPaint(HW\_wnd,&ps));  if(((GetDeviceCaps(ps.hdc,PLANES))!=1)||  ((GetDeviceCaps(ps.hdc,BITSPIXEL))!=sizeof(HW\_pixel)\*8)  )  //HW\_error("%d",(GetDeviceCaps(ps.hdc,BITSPIXEL)));  HW\_error("(Hardware) I'd rather have %d bit screen.",  sizeof(HW\_pixel)\*8  );  HW\_bmp=CreateCompatibleBitmap(ps.hdc,HW\_SCREEN\_X\_SIZE,HW\_SCREEN\_Y\_SIZE);  SelectObject(HW\_mem,HW\_bmp);  EndPaint(HW\_wnd,&ps);  HW\_rect.left=HW\_rect.top=0;  //窗口的大小  HW\_rect.right=HW\_SCREEN\_X\_SIZE;  HW\_rect.bottom=HW\_SCREEN\_Y\_SIZE;  ShowWindow(HW\_wnd,HW\_cmd\_show); /\* generate messages \*/  UpdateWindow(HW\_wnd);  }  //复制图像位图到物理设备存储器  void HW\_blit(void)  {  PAINTSTRUCT ps;  //  BeginPaint(HW\_wnd,&ps);  SelectPalette(ps.hdc,HW\_palette,FALSE);  RealizePalette(ps.hdc);  SetMapMode(ps.hdc,MM\_TEXT);  SetBitmapBits(HW\_bmp,G\_c\_buffer\_size\*sizeof(HW\_pixel),G\_c\_buffer);  BitBlt(ps.hdc,0,0,HW\_SCREEN\_X\_SIZE,HW\_SCREEN\_Y\_SIZE,HW\_mem,0,0,SRCCOPY);  EndPaint(HW\_wnd,&ps);  }  void app\_main(void) /\* rendering loop \*/  {  //光线跟踪场景中设置摄像机  TR\_set\_camera(0,0,500, 0,0,0, 1,0,0, 0,1,0);  //TR\_set\_camera(0,0,900, 0,0,0, 1,0,0, 0,1,0);  //关键中的关键，光线跟踪窗口中的pixel  TR\_trace\_world(w,10);  G\_text(10,10,"<enter-exit>", CL\_colour(CL\_COLOUR\_LEVELS-1,CL\_COLOUR\_LEVELS-1,CL\_COLOUR\_LEVELS-1),  CL\_LIGHT\_LEVELS-1,CL\_LIGHT\_LEVELS-1,CL\_LIGHT\_LEVELS-1);  HW\_blit();  }  void app\_handler(int kk) /\* event handler \*/  {  HW\_close\_event\_loop();  }  void HW\_close\_screen(void)  {  DeleteDC(HW\_mem);  DeleteObject(HW\_bmp);  }  int main(int n, char \*\*o)  {  char \*display;  if (n == 2)  display = o[1];  else  display = NULL;  strcat(path,"tracer.dat");  w = (struct TR\_world \*)D\_data(path);  CL\_init\_colour();  TR\_init\_rendering(TR\_SPECULAR|TR\_SHADOW|TR\_REFLECT);  TR\_init\_world(w);  G\_init\_graphics();  HW\_init\_screen(display,"3D RayTracing");  HW\_init\_event\_loop(app\_main,app\_handler);  HW\_close\_screen();  return(1);  }  //事件循环函数，第一个函数指针被不停执行，第二个响应外部时间  void HW\_init\_event\_loop(void (\*application\_main)(void),  void (\*application\_key\_handler)(int key\_code)  )  {  MSG msg;  HW\_application\_main=application\_main;  HW\_application\_key\_handler=application\_key\_handler;  while(1)  {  if(PeekMessage(&msg,NULL,0,0,PM\_REMOVE)) /\* this IS sensitive part! \*/  {  if(msg.message == WM\_QUIT) break;  TranslateMessage(&msg);  DispatchMessage(&msg);  }  else  {  InvalidateRect(HW\_wnd,&HW\_rect,TRUE);  UpdateWindow(HW\_wnd);  }  }  }   1. 运行结果截图： | | | |
| 分析与结论：  光线追踪，简单地说，就是从摄影机的位置，通过影像平面上的像素位置(比较正确的说法是取样(sampling)位置)，发射一束光线到场景，求光线和几何图形间最近的交点，再求该交点的著色。如果该交点的材质是反射性的，可以在该交点向反射方向继续追踪。光线追踪除了容易支持一些全局光照效果外，亦不局限于三角形作为几何图形的单位。任何几何图形，能与一束光线计算交点(intersection point)，就能支持。这个方法根据给定的射线可以返回一个整数表示与射线有无交点，和交点到射线原点的距离。总之，光线追踪就是一个不断递归计算的过程，指导光线强度低于一定的阈值时停止。这是三个实验里面比较综合的了，不过通过这个实验，我对计算机图形学的掌握程度又有了一个新的提高，为以后的继续学习打下了一个基础。 | | | |