A7: Supersampling and Antialiasing

一、概述

**1.目标：**

（1）为我们的ray tracer改善失真（抗锯齿）

（2）通过超级采样和过滤的方法实现

（3）实验各种采样方式和过滤方式

**2.知识点介绍：**

1. 为什么会出现失真（锯齿）？

物体的颜色是连续的，但是计算机图像是离散的，是由像素构成的。从物体的颜色到像素默认采用的中心采样法，即每个像素的中心点颜色决定了整个像素的颜色。这样就可能会丢失物体的细节，产生锯齿。

1. 如何实现抗锯齿？

增加采样点，采用不同的采样方式，获取该像素周围的颜色信息。然后根据所有获得的颜色信息计算最后的颜色。具体实现方式是：将每个像素的采样值存在film类中，filter类根据存储的采样颜色来计算最终颜色。

二、实现细节

1. **采样方式**

* 随机采样：随机取一个0-1之内的浮点数

|  |
| --- |
| Vec2f RandomSampler::getSamplePosition (int n) {  return Vec2f (static\_cast<float>(rand () % 10000 / 10000.0), static\_cast<float>(rand () % 10000 / 10000.0));} |

* 均一采样：按照给定的采样点数，将像素平分

|  |
| --- |
| Vec2f UniformSampler::getSamplePosition (int n) {  \_step = int (sqrtf (this->nSamples));  int x = n % \_alignedStep, y = n / \_step;  float xPos=1.f /\_step \*(x+0.5f),yPos = 1.f / \_step \* (y + 0.5f);  return Vec2f (xPos, yPos);} |

* 抖动采样：在均一采样的基础上，再加入一个随机的浮动值

|  |
| --- |
| Vec2f JitteredSampler::getSamplePosition (int n) {  Vec2f uniformPos = UniformSampler::getSamplePosition (n);  Vec2f offset = Vec2f ( (rand()% 10000 / 10000.0) - 0.5f, ((rand()%10000 / 10000.0) - 0.5f) / float (\_step));  Vec2f jitteredPos; Vec2f::Add (jitteredPos, uniformPos, offset);  jitteredPos.Clamp (); return jitteredPos;} |

1. **过滤方式**

* Box：坐标在半径之内的权重为1

|  |
| --- |
| float BoxFilter::getWeight (float x, float y) {  return fabsf (x) < radius && fabsf (y) < radius;} |

* Tent：在半径之内，采样点离中心越远，权重越低

|  |
| --- |
| float TentFilter::getWeight (float x, float y) {  float dist = sqrtf (x \* x + y \* y);  return max (0.f, 1.f - dist / radius);} |

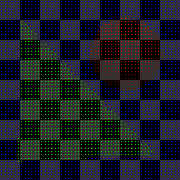
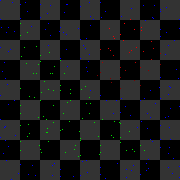
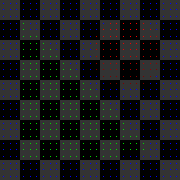
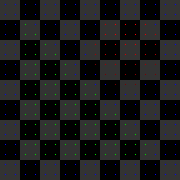
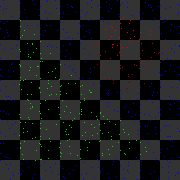
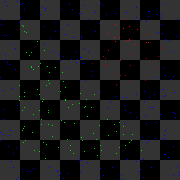
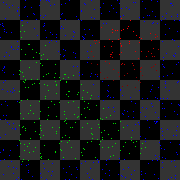
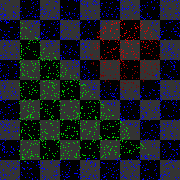
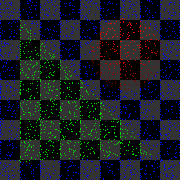
* Gaussian：采样点的权重在半径之内符合正态分布

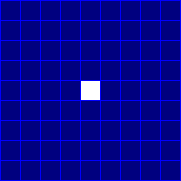
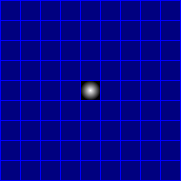
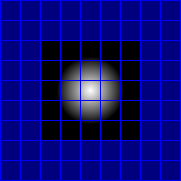
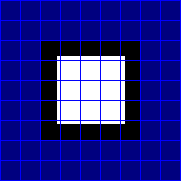
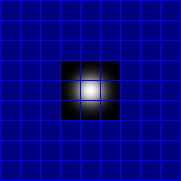
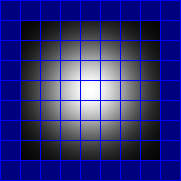
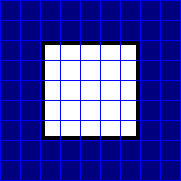
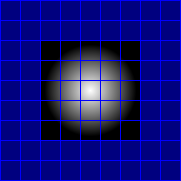
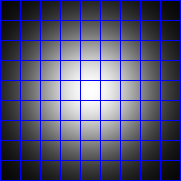
|  |
| --- |
| float GaussianFilter::getWeight (float x, float y) {  float dist = sqrtf (x \* x + y \* y);  return expf ((-dist \* dist) / (2.f \* sigma \* sigma));} |

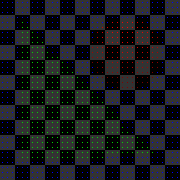
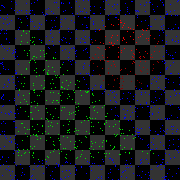
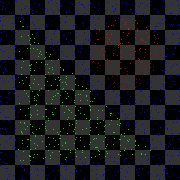
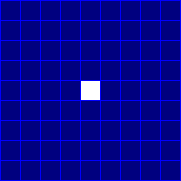
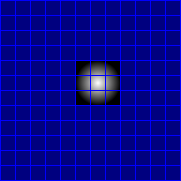
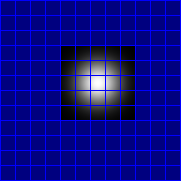
* 计算最后颜色的方法：将所有采样点乘以权重的颜色加起来

|  |
| --- |
| Vec3f Filter::getColor (int i, int j, Film \*film) {  Vec3f color; float sumWeight = 0.0f;  int supportRadius = getSupportRadius ();  for (int xOffset = -supportRadius; xOffset <= supportRadius; ++xOffset) {  for (int yOffset = -supportRadius; yOffset <= supportRadius; ++yOffset) {  int x = xOffset + i, y = yOffset + j;  if (x<0||x>=film->getWidth()||y<0||y>=film->getHeight()) continue;  for (int n = 0; n < film->getNumSamples (); ++n) {  Sample sample = film->getSample (x, y, n);  Vec2f samplePos = sample.getPosition ();  Vec2f pixelPos = Vec2f (samplePos.x () + xOffset - 0.5f, samplePos.y () + yOffset - 0.5f);  float weight = getWeight (pixelPos.x (), pixelPos.y ()); color += sample.getColor () \* weight;  }  } }  return color;  } |

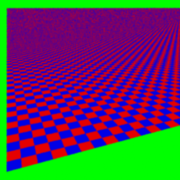
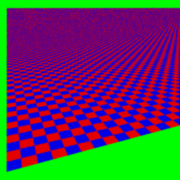
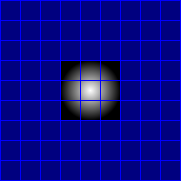
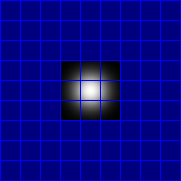
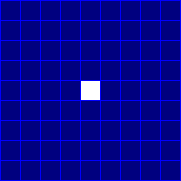
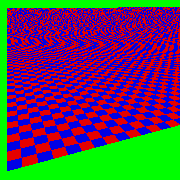
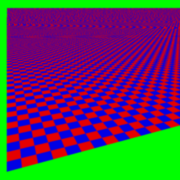
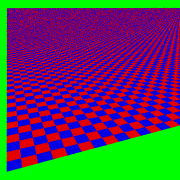
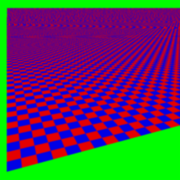
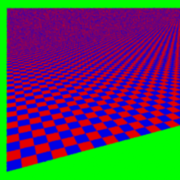
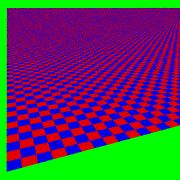
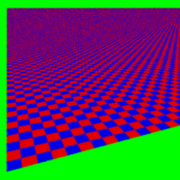
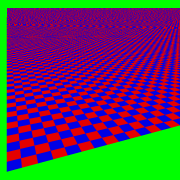
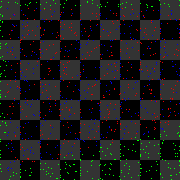
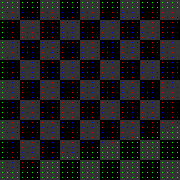
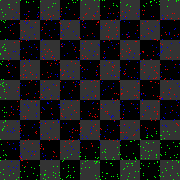
三、结果展示

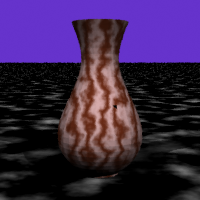
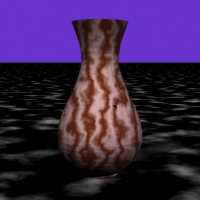
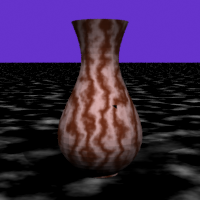
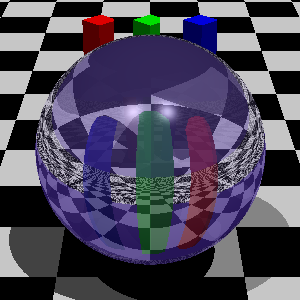
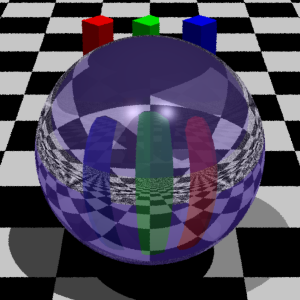
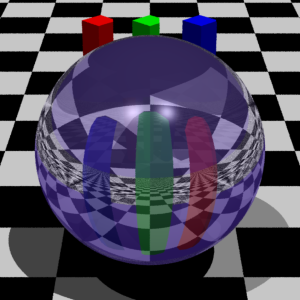
1.采样测试：

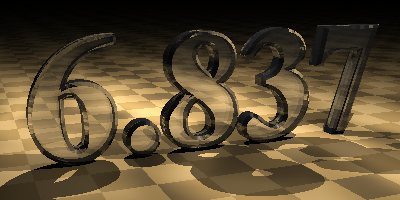
2.过滤测试：

E:\MyStudy\Assignments\Assignments\Debug\A6\Png7\output7_01i_low_res - 副本.pngE:\MyStudy\Assignments\Assignments\Debug\A6\Png7\output7_01h_low_res - 副本.pngE:\MyStudy\Assignments\Assignments\Debug\A6\Png7\output7_01g_low_res - 副本.pngE:\MyStudy\Assignments\Assignments\Debug\A6\Png7\output7_01f_low_res - 副本.pngE:\MyStudy\Assignments\Assignments\Debug\A6\Png7\output7_01e_low_res - 副本.pngE:\MyStudy\Assignments\Assignments\Debug\A6\Png7\output7_01d_low_res - 副本.pngE:\MyStudy\Assignments\Assignments\Debug\A6\Png7\output7_01c_low_res - 副本.pngE:\MyStudy\Assignments\Assignments\Debug\A6\Png7\output7_01b_low_res - 副本.pngE:\MyStudy\Assignments\Assignments\Debug\A6\Png7\output7_01a_low_res - 副本.pngE:\MyStudy\Assignments\Assignments\Debug\A6\Png7\output7_01_low_res - 副本.png3.不同采样和过滤方式组合测试：

综合以上结果来看：对于超级采样减轻失真的效果，在采样点数目相同和过滤方式相同的情况下，jittered\_samples效果最好。而过滤方式上：gaussian>tent>box。此外当然，采样点越多效果越好。高斯过滤虽然减轻了失真，但是可以看到整个画面变得有些模糊。如果想要获得好的减轻失真的效果，可以选择jittered采样和高斯过滤，提高采样点数量，之后的实验中也是这样做的。

4.减轻失真测试：花瓶这里由于环境光没有做好，差别不是很明显。



四、心得体会

通过增加采样点，改变采样方式和用不同的filter来计算最终颜色，可以有效地提高画面效果，减轻失真。但是同时也加大了计算量，光线投射的量成倍增长，渲染速度变慢（让人不禁想起在游戏中开启抗锯齿效果之后会变得有些卡）。