**דוח**

שיפורים שעשינו:

-Glossy/ Diffuse

-Adaptive Supersampling

**שיפור glossy/ Diffuse**

הבעיה הייתה שההחזא של הגופים על שטח glossy היה יותר מדי מדויק וזה לא דומה למציאות. גם לגוף שהוא מאחורי משהו שקוף אז אמורים לראות אותו קצת מטושטש.

רעיון של השיפור הוא לשלוח אלומת קרניים מסביב לקרן הראשית (שהוא או קרן השתקפות או קרן שקיפות) כדי לקבל את הטשטוש הרצוי.

private Color calcSampledColor(double radius, int level, Ray refRay, Vector n, double k) {  
  
  
 Vector v=refRay.get\_dir();*//director vector of the reflected or refracted ray* Point3D point=refRay.get\_p0();*// p0 of the reflected or refracted ray* GeoPoint gp = findClosestIntersection(refRay);*// gp is the closest GeoPoint intersection with the ray* Color color=Color.*BLACK*;  
 double ndir=*alignZero*(v.dotProduct(n));  
 if(ndir==0) {*// if the director vector of the ray and the normal are perpendicular* return color;*// return black* }  
 Color bg = \_scene.background;*// bg is the color of the background* if(gp==null)  
 color=bg;*//if there is no intersection we return the background color* else  
  
 color = calcColor(gp, refRay, level - 1, k);*// color of gp* Ray r= new Ray(point,n, v);*// r is new version of reflected or refracted ray (with delta)* if(this.getSampleCount()==0)  
 return color;  
  
 double x = v.getHead().getX();  
 double y = v.getHead().getY();  
 double z =v.getHead().getZ();  
  
 Vector u = null;*// u is perpendicular to vector v (director vector of reflected or refracted ray)* if (x <= y) {  
 if (x <= z)  
 u = new Vector(0, z, -y).normalize();  
 else  
 u = new Vector(-y, x, 0).normalize();  
 } else {  
 if (y <= z)  
 u = new Vector(-z, 0, x).normalize();  
 else  
 u = new Vector(-y, x, 0).normalize();  
 }  
  
 Vector w= u.crossProduct(refRay.get\_dir()).normalize();*// w is perpendicular to v and u* for (int i=0;i<this.getSampleCount()-1;i++) {  
 double nw;  
 Point3D p0;  
 Point3D pt;  
 Vector w1;  
 do {  
 double cosTeta = *RANDOM*.nextDouble()\*2-1;  
 double sinTeta = Math.*sqrt*(1 - cosTeta \* cosTeta);  
  
 p0 = r.get\_p0();  
 double distance = r.get\_dir().length();*//length of refected or refracted ray* Point3D pc = p0.add(r.get\_dir().scale(distance));*// pc is point at the end of the reflected or refracted ray* pt = pc;  
 if (!*isZero*(cosTeta))  
 pt = pt.add(u.scale(cosTeta));*// we move pc point of cosTeta on u axis* if (!*isZero*(sinTeta))  
 pt = pt.add(w.scale(sinTeta));*//we move pc point of sinteta on w axis* z=radius\*(*RANDOM*.nextDouble()\*2-1);  
 pt=pc.add(pt.subtract(pc).scale(z));*//we move pc point of z* nw= *alignZero*(n.dotProduct(pt.subtract(refRay.get\_p0())));*// nw is dotproduction between the normal and the vector between pc and the new point* } while (nw<0&&ndir>0 ||ndir<0&&nw>0);  
  
 Ray ray=new Ray(p0,pt.subtract(p0));*// ray is the new ray in the beam of ray* gp=findClosestIntersection(ray);  
 color=color.add(gp==null?bg:calcColor(gp,ray,level-1,k));*// we add to color the calculated color of gp* }  
  
 return color.reduce(this.getSampleCount()+1);*// the average of all colors*}

Graphical user interface, application

Description automatically generatedלפני השיפור: אחרי השיפור:

A screenshot of a computer

Description automatically generated with medium confidence

**:Adaptive Supersampling שיפור**

הבעייה הייתה שהתמטנה לא הייתה מדויקת. אז הפתרון הוא לשלוח יותר קרניים דרך אותו פיקסל.(זה העיון של .(supersampling

Adaptive Supersampling הוא שיפור של סופרסמפלינג. הרעיון הוא לחלק את הפיקסל רק במקומות שבהן הצבעים שונים. עוצרים את הרקורסיה רק כשהצבעים דומים או שרמת הרקורסיה (ה-LEVEL) שווה לאפס.

אם רוצים לבצע את השיפור הזה צריך לקרוא את הפונקציה הבאה:

render.setLevel\_adaptive\_supersampling(LEVEL);

בפונקציה מאתחלים את ה-LEVEL כי ברירת המחדל ה-LEVEL שווה לאפס.

אם לא קוראים לפונקציה הזו אז לא מבצעים את השיפור הזה. כל הפונקציות הקשורות לשיפור הזה נמצאות במחלקת רנדר.

int level\_adaptive\_supersampling=0;//LEVEL  
  
public void setLevel\_adaptive\_supersampling(int level) {  
 level\_adaptive\_supersampling = level;  
  
}

public void renderImage() {  
 if (\_imageWriter == null)  
 throw new MissingResourceException(*RESOURCE\_ERROR*, *RENDER\_CLASS*, *IMAGE\_WRITER\_COMPONENT*);  
 if (\_camera == null)  
 throw new MissingResourceException(*RESOURCE\_ERROR*, *RENDER\_CLASS*, *CAMERA\_COMPONENT*);  
 if (\_rayTracerBase== null)  
 throw new MissingResourceException(*RESOURCE\_ERROR*, *RENDER\_CLASS*, *RAY\_TRACER\_COMPONENT*);  
 Color pixelColor=Color.*BLACK*;  
 final int nX = \_imageWriter.getNx();  
 final int nY = \_imageWriter.getNy();  
 if (threadsCount == 0)  
 for (int i = 0; i < nY; ++i)  
 for (int j = 0; j < nX; ++j)  
 if(level\_adaptive\_supersampling==0)  
 castRay(nX, nY, j, i);  
 else {  
  
 pixelColor = AdpativeSuperSampling(nX, nY, i, j, \_rayTracerBase);  
 \_imageWriter.writePixel(j, i, pixelColor);  
 }  
 else  
 renderImageThreaded();  
}

הפונקציה 1 היא זאת שמחשבת את הנקודות של 4 הפינות של הפיקסל. היא קוראת לפונקציה 2 שהיא מייצרת את הקרניים של 4 הפינותת מחשבת את הצבעים. היא בודקת שהצבעים דומים אם כן היא מחזירה את הממוצע של הצבעים. אם לא היא קוראת לפונקציה 3. הפונקציה 3 היא הפונקציה הרקורסיבית שהיא מחלקת את הפיקסל ל-4 תתי פקסלים. היא עוברת בינהם ואז ממשיכה לחלק אם יש צורך עד שאחד התנאים של סוף הרקורסיה (צבעים דומים או LEVEL שווה לאפס) ייתקיים.

1

public Color AdpativeSuperSampling(int nX, int nY, int i, int j, RayTracerBase rayTracerBase) {  
  
 double Ry = \_camera.get\_height() / nY;*// Ry is length of pixel* double Rx = \_camera.get\_width() / nX; *//Rx is width of pixel* Point3D Pc = \_camera.getP0().add(\_camera.getvTo().scale(\_camera.get\_Distance()));*// Pc is the interscetion point on the center of the viewplane* Point3D PcornerLT = Pc.add(\_camera.getvRight().scale(-\_camera.get\_height() / 2).add(\_camera.getvUp().scale(\_camera.get\_width() / 2))); *//PcornerLT is the point on the top corner left of the viewplane* Point3D PLT;  
  
 if (j == 0 && i == 0)*//if the current pixel is the first pixel at the top corner left* PLT = PcornerLT;  
 else {  
 if (j == 0 && i != 0)*// if the current pixel is in the first column* PLT = PcornerLT.add(\_camera.getvUp().scale(-Rx \* i));  
 else if (i == 0 && j != 0)*//if the current pixel is in the first row* PLT = PcornerLT.add(\_camera.getvRight().scale(Ry \* j));  
 else  
 PLT = PcornerLT.add(\_camera.getvUp().scale(-Rx \* i)).add(\_camera.getvRight().scale(Ry \* j));*// if the current pixel is other* }  
 Point3D PRT = PLT.add(\_camera.getvRight().scale(Ry));*// PRT is the point at the top right of the current pixel* Point3D PBR = PLT.add(\_camera.getvRight().scale(Ry).add(\_camera.getvTo().scale(-Rx)));*// PBR is the point at the bottom right of the current pixel* Point3D PBL = PLT.add(\_camera.getvUp().scale(-Rx));*// PBL is the point at the bottom left of the current pixel* Color color;  
 color = AdpativeSuperSamplingCalcCol(Rx, Ry, PLT, PRT, PBR, PBL, rayTracerBase);  
 return color;  
  
  
}

2

public Color AdpativeSuperSamplingCalcCol(double Rx, double Ry, Point3D p0, Point3D p1, Point3D p2, Point3D p3, RayTracerBase rayTracerBase) {  
 Ray raylt = new Ray(\_camera.getP0(), p0.subtract(\_camera.getP0()));*//* Color clt = rayTracerBase.traceRay(raylt);  
 Ray rayrt = new Ray(\_camera.getP0(), p1.subtract(\_camera.getP0()));  
 Color crt = rayTracerBase.traceRay(rayrt);  
 Ray rayrb = new Ray(\_camera.getP0(), p2.subtract(\_camera.getP0()));  
 Color crb = rayTracerBase.traceRay(rayrb);  
 Ray raylb = new Ray(\_camera.getP0(), p3.subtract(\_camera.getP0()));  
 Color clb = rayTracerBase.traceRay(raylb);  
 Color color=Color.*BLACK*;  
  
 if (clt.equal(crt) && crt.equal(crb) && crb.equal(clb))*// if the 4 corners of the current pixel have a similar color return the average of the 4 colors* return color.add(clt,crt,crb,clb).reduce(4);  
  
 int level = level\_adaptive\_supersampling;  
 Color color1 = Color.*BLACK*;  
 color = AdaptiveSuperSamplingRecursive(Rx, Ry, p0, p1, p2, p3, rayTracerBase, clt, crt, crb, clb, level, color1);  
 return color;  
  
}

3  
  
public Color AdaptiveSuperSamplingRecursive(double Rx, double Ry, Point3D p0, Point3D p1, Point3D p2, Point3D p3, RayTracerBase rayTracerBase, Color c0, Color c1, Color c2, Color c3, int level, */\*double prop,\*/* Color color1) {  
 if (level == 0) *//if level=0 we stop the recursive and we return the color* return color1;  
 Rx = Rx / 2; *//Rx is the half the width of the pixel* Ry = Ry / 2;*//Ry is the half the length of the pixel* Point3D p5 = p0.add(\_camera.getvRight().scale(Ry)); *//p5 is the midpoint between the top left point and the the top right point* Point3D p6 = p1.add(\_camera.getvUp().scale(-Rx));*//p6 is the midpoint between the top right point and the the bottom right point* Point3D p7 = p2.add(\_camera.getvRight().scale(-Ry));*//p7 is the midpoint between the bottom right point and the the bottom left point* Point3D p8 = p0.add(\_camera.getvUp().scale(-Rx));*//p8 is the midpoint between the bottom left point and the the top left point* Point3D p4 = p0.add(\_camera.getvRight().scale(Ry).add(\_camera.getvUp().scale(-Rx)));*//p4 is the point in the center of the pixel* Ray ray5 = new Ray(\_camera.getP0(), p5.subtract(\_camera.getP0()));*//ray5 is the ray that passes through p5* Color c5 = rayTracerBase.traceRay(ray5); *//c5 is the color of ray5* Ray ray6 = new Ray(\_camera.getP0(), p6.subtract(\_camera.getP0()));  
 Color c6 = rayTracerBase.traceRay(ray6);  
 Ray ray7 = new Ray(\_camera.getP0(), p7.subtract(\_camera.getP0()));  
 Color c7 = rayTracerBase.traceRay(ray7);  
 Ray ray8 = new Ray(\_camera.getP0(), p8.subtract(\_camera.getP0()));  
 Color c8 = rayTracerBase.traceRay(ray8);  
 Ray ray4 = new Ray(\_camera.getP0(), p4.subtract(\_camera.getP0()));  
 Color c4 = rayTracerBase.traceRay(ray4);  
  
 *//we pass on the 4 sub pixels formed* if (c0.equal(c5) && c5.equal(c4) && c4.equal(c8)) *//if the 4 colors of the top left sub pixel are similar we return the average of the 4 colors* {  
 color1 = color1.add(c0,c5,c4,c8);  
 color1=color1.reduce(4);}  
 else {  
 if (level - 1 == 0) { *//if the next time we call the function is the last time as we calculate the final color of the sub pixel* color1 = c0.add(c5).add(c4).add(c8);  
 color1 = color1.reduce(4);  
 }  
  
  
 color1 = color1.add(AdaptiveSuperSamplingRecursive(Rx, Ry, p0, p5, p4, p8, rayTracerBase, c0, c5, c4, c8, level - 1,color1)).reduce(2);  
  
 }  
  
 if (c5.equal(c1) && c1.equal(c6) && c6.equal(c4)) {*//if the 4 colors of the top right sub pixel are similar we return the average of the 4 colors* color1 = color1.add(c5,c1,c6,c4);  
 color1 = color1.reduce(5);  
 } else {  
 if (level - 1 == 0) {  
 color1 = c5.add(c1).add(c6).add(c4);  
 color1 = color1.reduce(4);  
 }  
  
 color1 = color1.add(AdaptiveSuperSamplingRecursive(Rx, Ry, p5, p1, p6, p4, rayTracerBase, c5, c1, c6, c4, level - 1,color1)).reduce(2);  
  
 }  
 if (c4.equal(c6) && c6.equal(c2) && c2.equal(c7)) {*//if the 4 colors of the bottom right sub pixel are similar we return the average of the 4 colors* color1 = color1.add(c4,c6,c2,c7);  
 color1 = color1.reduce(5);  
 } else {  
 if (level - 1 == 0) {  
 color1 = c4.add(c6).add(c2).add(c7);  
 color1 = color1.reduce(4);  
 }  
  
 color1 = color1.add(AdaptiveSuperSamplingRecursive(Rx, Ry, p4, p6, p2, p7, rayTracerBase, c4, c6, c2, c7, level - 1, color1)).reduce(2);  
 }  
 if (c8.equal(c4) && c4.equal(c7) && c7.equal(c3))*//if the 4 colors of the bottom left sub pixel are similar we return the average of the 4 colors* {  
 color1 = color1.add(c8,c4,c7,c3);  
 color1=color1.reduce(5);}  
 else {  
 if (level - 1 == 0) {  
 color1 = c8.add(c4).add(c7).add(c3);  
 color1=color1.reduce(4);}  
 color1 = color1.add(AdaptiveSuperSamplingRecursive(Rx, Ry, p8, p4, p7, p3, rayTracerBase, c8, c4, c7, c3, level - 1,color1)).reduce(2);  
 }  
 return color1;  
}

לפני השיפור: אחרי השיפור:

A screenshot of a computer

Description automatically generated with medium confidenceA screenshot of a computer

Description automatically generated with medium confidence

**תהליכונים:**

לקחנו את הקוד של דן זילברשטיין שבמודל והתאמנו אותו לקוד שלנו.

יצרנו סנצה ראשית. יצירת התמונה עם השיפור של גלוסי ודיפיוז בלי תהליכונים לקחה 8 שעות 19 דקות 37 שניות. ויצירת התמונה עם השיפור של גלוסי ודיפיוז עם תהליכונים לקחה 5 שעות 58 דקות 45 שניות.

**סצנות:**

סצנה 1:

תמונה עם כל הגופים לפני שיפורים:

Graphical user interface, application

Description automatically generated

לפני כל השיפורים : אחרי adaptive supersampling

Graphical user interface

Description automatically generatedGraphical user interface

Description automatically generated

אחרי גלוסי :

Graphical user interface, application

Description automatically generated