מיני פרויקט 2- שיפור ביצועים

מגישות: אביה אלפסי
סימא פיליפס

הבעיה:

זמן הריצה של יצירת תמונה הוא ארוך מאד. ככל שיש יותר גופים בתמונה הוא מתארך כי יש יותר נקודות חיתוך של קרניים על הגופים ויותר קריאות לפונקציה הרקורסיבית calcColor, במיוחד לאחר שעשינו Super Sampling ומספר הקרניים גדל מאד.

הפתרון:

א. תהליכונים:

השתמשנו ב-Multi Threading, שימוש בתהליכונים מרובים, זה מקצר את זמן הריצה כיוון שדברים נעשים "במקביל". השתמשנו ב-3 תהליכונים לכל היותר.
הוספנו למחלקה Render משתנה threads ששומר את מספר התהליכונים הרצויים, ומשתנה SPARE_THREADS ששומר כמה תהליכונים פנויים יש.
כמו כן, הוספנו מחלקת עזר חדשה של Pixel שיש לה גישה לכל המשתנים הפנימיים ב-Render.
הבחירה להוסיף את המחלקה היתה מתוך שיקולים של ניהול וסדר בפרויקט. היא אפשרה לקיים אובייקט יחיד שמנהל את כל ענייני הפיקסל, המעבר לפיקסל הבא, הבדיקה שנותרו עוד פיקסלים, שמירת הפיקסל הנוכחי וכו'.
הפעולה שנעשתה על ידי כל תהליכון: התהליכון מקבל את הפיקסל הבא בתור לעיבוד. בתוך הפונקציה RenderImage מופעל התהליכון. בכל תהליכון נבדק האם יש צורך בשימוש בפונקציות של השיפורים (Super Sampling, Adaptive Super Sampling) ובהתאם נעשה החישוב המתאים של צבע הפיקסל. ברגע שלא נותרו עוד פיקסלים בתור התהליכון מסתיים.

ב. Adaptive Super Sampling:

הרעיון של Adaptive Super Sampling הוא לחלק כל פיקסל לאיזורים ולבדוק כל איזור, שולחים תחילה קרן אחת דרך כל פינה ואמצע הפיקסל- אם כל צבעי הקרניים זהות נחזיר את הצבע של הקרן שעוברת דרך מרכז הפיקסל. אם יש קרן בעלת צבע שונה נבדוק שוב את האיזור סביבה, קטע קטן יותר של הפיקסל, באותה דרך שבדקנו לפני- השוואת 4 פינות מסביב+אמצע, וחוזר חלילה, ברקורסיה, כך שנעבור על כל הפיקסל אם יש צורך ונקבל את הצבע המדויק, אך אם אין צורך בקרניים מרובות דרך פיקסל יחיד, נחסך ממנו זמן ריצה רב.
הוספנו את הפונקציה הבאה, ששולחת את הפינות של הפיקסל אל פונקציה נוספת, רקורסיבית, שעושה את החישוב של הצבע בשימוש ב-Adaptive Super Sampling:

```
private  Color adaptiveSuperSampling(Camera camera, double screenDistance, int nx, int ny, double width, double height, int col, int row) {
    List<Ray> rays = camera.constructRaysThroughPixel(nx, ny, col, row, screenDistance, width, height, NUM_OF_SAMPLE_RAYS); // calculate the rays throw the pixel
    int ray1Index = (NUM_OF_SAMPLE_RAYS-1)*NUM_OF_SAMPLE_RAYS+(NUM_OF_SAMPLE_RAYS-1); //the index of the up and right ray
    int ray2Index =(NUM_OF_SAMPLE_RAYS-1)*NUM_OF_SAMPLE_RAYS;  //the index of the up and left ray
    int ray3Index= 0;  //the index of the button and left ray
    int ray4Index = (NUM_OF_SAMPLE_RAYS-1);  // the index of the button and right ray

    Color color = adaptiveSuperSampling(rays, LEVEL_OF_ADAPTIVE, ray1Index, ray2Index, ray3Index, ray4Index); //calculate the color for the pixel
    return color;
}
```

הפונקציה הרקורסיבית:

```
private Color adaptiveSuperSampling(List<Ray> rays, int level_of_adaptive,int ray1Index, int ray2Index, int ray3Index, int ray4Index) {
    int numOfAdaptiveRays = 5;

    Ray centerRay = rays.get(rays.size()-1); //get the center screen ray
    Color centerColor=calcColor(centerRay); //get the color of the center
    Ray ray1 = rays.get(ray1Index);  //get the up and right screen ray
    Color color1=calcColor(ray1); //get the color of the up and right
    Ray ray2 = rays.get(ray2Index); //get the up and left ray
    Color color2=calcColor(ray2);//get the color of the up and left
    Ray ray3 = rays.get(ray3Index);//get the button and left ray
    Color color3=calcColor(ray3);//get the color of the button and left
    Ray ray4 = rays.get(ray4Index);//get the button and right ray
    Color color4=calcColor(ray4);//get the color of the button and right
    if (level_of_adaptive == 0)
    {
        //Calculate the average color of the corners and the center
        centerColor = centerColor.add(color1,color2, color3,color4);
        return centerColor.reduce(numOfAdaptiveRays);
    }
    //If the corner color is the same as the center color, returns the center color
    if (color1.isColorsEqual(centerColor) && color2.isColorsEqual(centerColor) && color3.isColorsEqual(centerColor) && color4.isColorsEqual(centerColor))
    {
        return centerColor;
    }
    //Otherwise, for each color that is different from the center, the recursion goes down to the depth of the pixel and sums up
    // the colors until it gets the same color as the center color,
    else {
        if (!color1.isColorsEqual(centerColor)) {
            color1 = color1.add(adaptiveSuperSampling(rays, level_of_adaptive: level_of_adaptive - 1, ray1Index: ray1Index - (NUM_OF_SAMPLE_RAYS+1), ray2Index, ray3Index, ray4Index
            color1 = color1.reduce(2);
        }
        if (!color2.isColorsEqual(centerColor)) {;
            color2 = color2.add(adaptiveSuperSampling(rays, level_of_adaptive: level_of_adaptive - 1,ray1Index, ray2Index: ray2Index-(NUM_OF_SAMPLE_RAYS-1), ray3Index, ray4Index));
            color2 = color2.reduce(2);
        }
        if (!color3.isColorsEqual(centerColor)) {
            color3 = color3.add(adaptiveSuperSampling(rays, level_of_adaptive: level_of_adaptive - 1,ray1Index, ray2Index, ray3Index: ray3Index+(NUM_OF_SAMPLE_RAYS+1), ray4Index));
            color3 = color3.reduce(2);
        }
        if (!color4.isColorsEqual(centerColor)) {
            color4 = color4.add(adaptiveSuperSampling(rays, level_of_adaptive: level_of_adaptive - 1,ray1Index, ray2Index, ray3Index, ray4Index: ray4Index+(NUM_OF_SAMPLE_RAYS-1)));
            color4 = color4.reduce(2);
        }
        //Calculate and return the average color
        centerColor = centerColor.add(color1, color2, color3, color4);
        return centerColor.reduce(numOfAdaptiveRays);

    }
```

התוצאות הסופיות:

זמן הריצה עם Super Sampling:

21 דקות ו-24 שניות

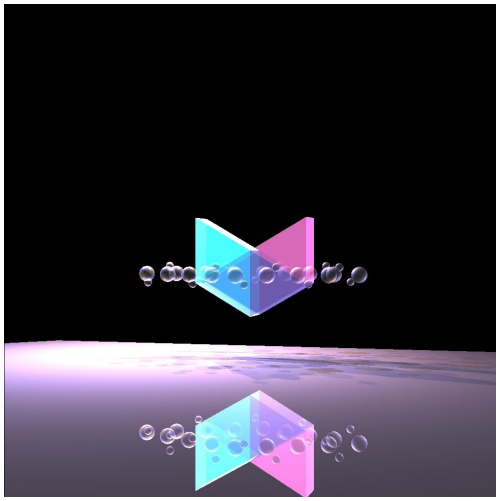
זמן הריצה עם Adaptive Super Sampling:

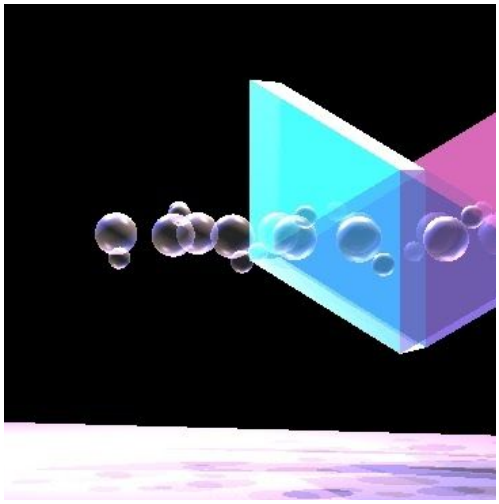
2 דקות ו-59 שניות

יש שיפור ניכר מאד ומשמעותי ביותר בזמן הריצה אחרי שהוספנו את ה-Adaptive Super Sampling

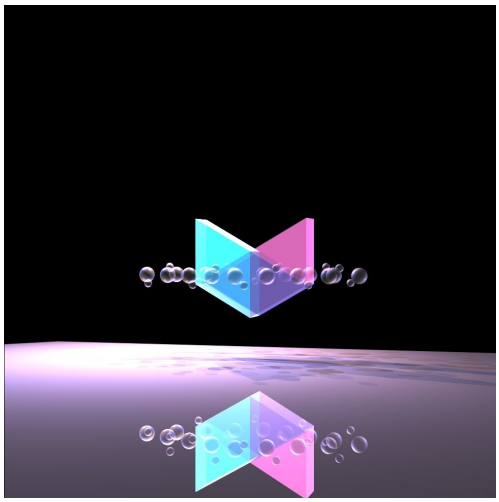
תוצאה סופית:

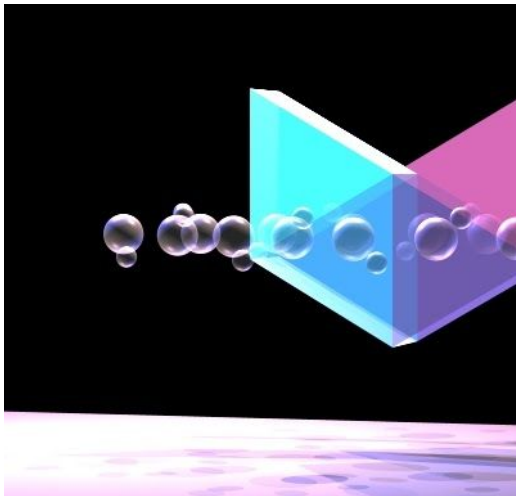
התמונה ללא שיפור:





התמונה לאחר שיפור Super Sampling רגיל:





התמונה לאחר שיפור Adaptive Super Sampling: