# 平时作业1报告

1. **概述**

本次作业目标为绘制一个由不同颜色的三角形面片组成的OpenGL六边形图像，共实现如下功能：

i. 搭建好OpenGL编程环境以及绘图窗口

ii. 为窗口背景指定初始颜色

iii. 实现绘制由近似色三角形面片组成的正六边形

iv. 实现快捷键C启动六边形中各三角形颜色渐变为另一色调

v. 实现快捷键S实现三角形边的高亮显示，同时所有三角形顶点出现大小不一

的圆点，每帧刷新亮点的大小和亮度，实现闪耀的视觉效果

vi. 实现线条上光斑移动以及闪烁

vii. 实现整个图像在画面上做圆周运动的功能

1. **工作流程**

本次作业历时4日，其中1~2日用于学习相关OpenGL知识，搭建环境，3~4日用于编写代码，并完成报告。

1. **技术方案**

①. **创建绘图窗口，并指定窗口基础颜色：**

在程序main函数内调用glfw库当中的glfwCreateWindow即可，并修改相关变量创建了一个1920x1080的全屏窗口，作为绘图窗口。在main函数的while循环当中，调用glClearColor，将函数参数的颜色设定为深蓝色即可。

②. **实现绘制由近似色三角形面片组成的正六边形：**

首先创建一个类：Point2D类，这个类代表了2D平面上的一个点，在该类当中除了实现一些基本的点乘、加、赋值、向量标准化函数，额外为项目创建了两个函数。

**函数1**：生成一个以窗口原点为中心，一边与y轴平行的正六边形点集，该函数接收一个参数length，代表从六边形中心到任意一个顶点的距离，返回由六个点组成的Point2D类型数组，代表一个正六边形的六个顶点。该函数实现的方式很简单，因为得到了原点到正六边形六个顶点的位置，只需要根据正六边形的图形特点返回六个顶点即可，此处不赘述。

**函数2**：向量旋转函数，接收一个角度值θ，返回将一个点（或者说向量）绕着z轴正方向逆时针旋转θ角度后得到的向量顶点值。该函数实现利用了基本的线性代数知识：

将一个二维向量绕原点逆时针旋转θ，等价为应用旋转矩阵：

首先利用函数1，获得六边形的六个顶点，再利用函数2将这个六边形逆时针旋转15°，接着为了获得更多的三角形面片，在六边形内部我额外设定了三个Inner Points（为一个等边三角形的三个点调用函数2进行一定的旋转获得）。这样就获得了组成三角形面片以及六边形所需的全部9个点。

接下来，因为绘制多个三角形面片不应该产生交叉线，所以为其编写了一个绘制顺序indices数组。接着为其绑定VAO、VBO、EBO，在最后while循环当中声明绘制方法为GL\_FILL，为每一个三角形调用glDrawArrays函数即可看见一个相同颜色的六边形（实际上是多个颜色一致的三角形面片拼成的）。

然后，在fragment shader文件当中定义一个uniform vec4变量baseColor，输出一个RGBA的颜色值，定义其有一个基础色（蓝色）：（0.0f, 0.1f, 0.4f, 0.0f），然后在绘制每一个三角形的时候，按照绘制顺序，给基础色稍微改变一下颜色值，得到相似色效果：

设定红色变量为：redBase + (i \* 0.035f)。

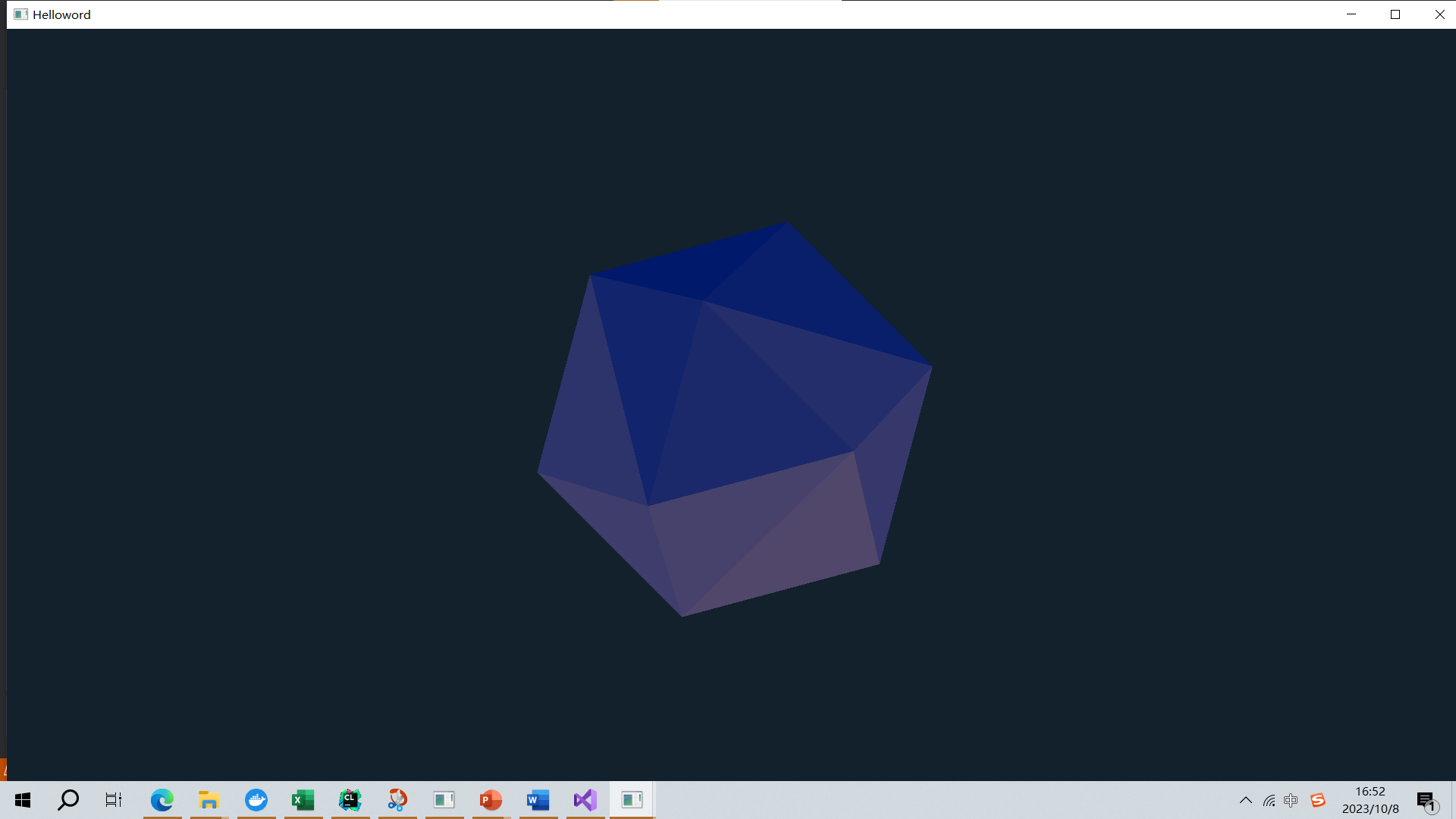
设定绿色变量为：greenBase + (i \* 0.02f)。

设定蓝色变量不变。

设定透明度变量为1.0f。

这样就完成了三角形显示相近色的功能。

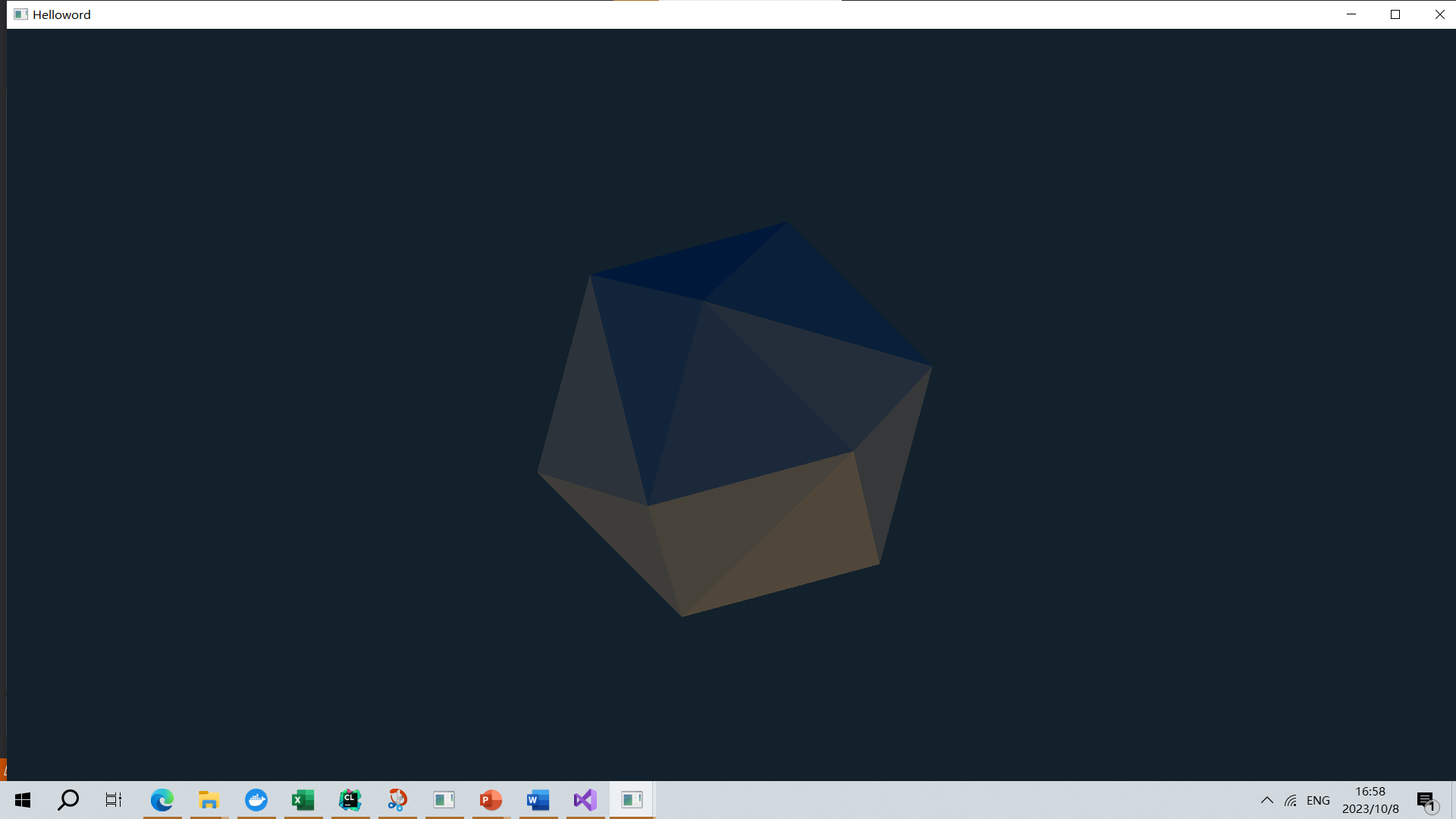
这一阶段程序启动，可以看见一个由多个面片组成的正六边形：



**③. 实现快捷键C启动三角形当中各个颜色渐变为另外一种颜色**

这个功能的实现方法是：将基础色的前三个值分别设置为redBase, greenBase, blueBase，当用户按下C键的时候，对这几个base值进行一定的周期性的更改，这里利用到了cos函数的周期性，当用户按下C键的时候，可以看见正六边形的颜色不断地改变，并且有一个循环。这样还有额外的好处：再次按C键取消渐变的时候，会保存当前渐变到的颜色，不会突然跳到原本的原色导致突兀的视觉效果。

这里我使用的更改方法为：blueBase = 0.42f + sin(colorChange) / 5.0f，对其余的颜色不进行更改，colorChange为一个在[0, 2π]内周期性均匀变化的值。这样就可以实现三角形的颜色渐变（以下图片是渐变到另外一种颜色后的画面效果）：



**④实现快捷键S实现三角形边的高亮显示，同时所有三角形顶点出现大小不一的圆点，每帧刷新亮点的大小和亮度，实现闪耀的视觉效果**

对于绘制三角形边，因为在第一步已经绑定好了VAO，只要在while循环当中进行绘制即可，此处不赘述。

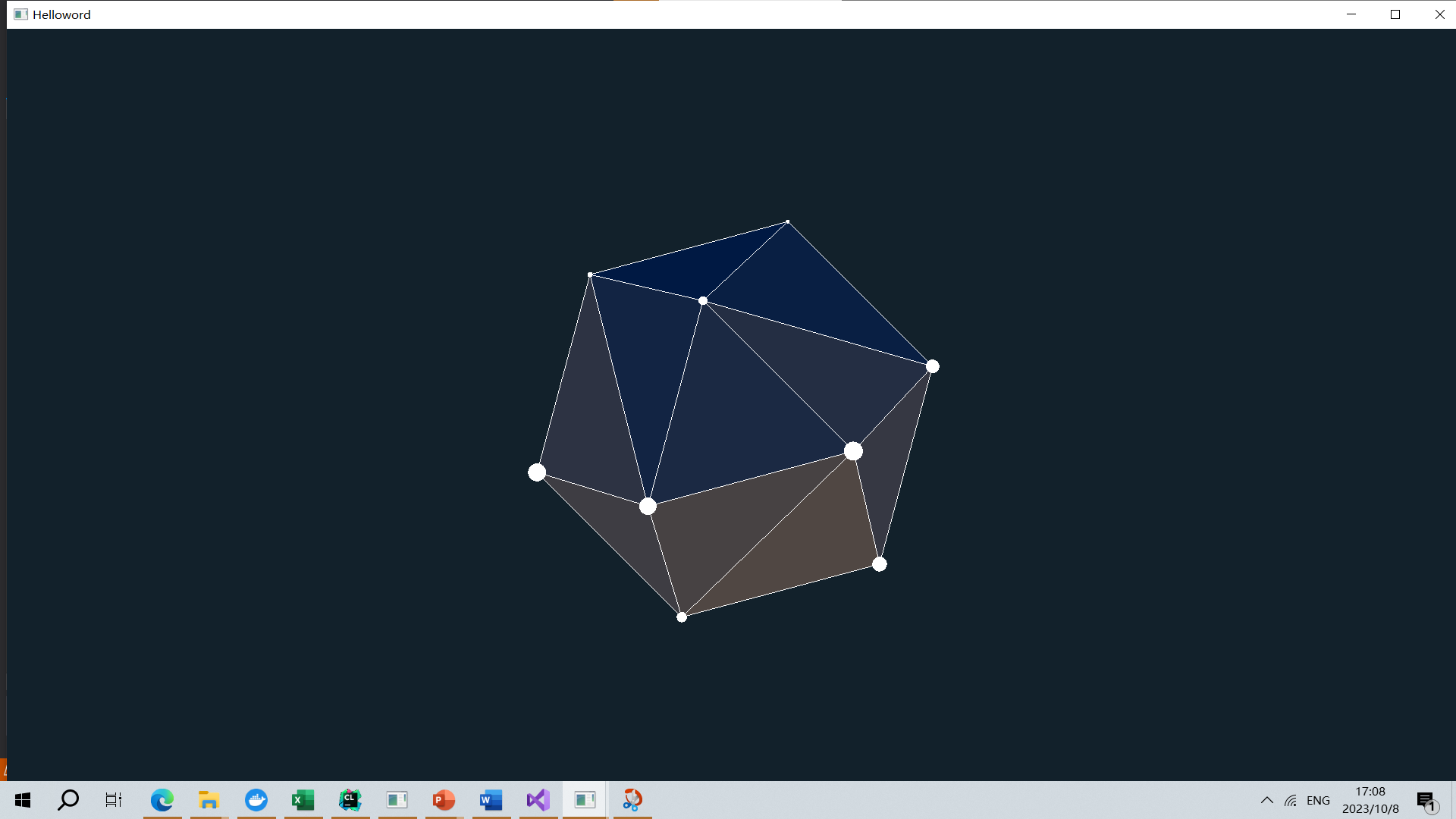
原本准备利用多边形来拟合一个圆的方法画出原点，但最后觉得这样过于复杂，而且浪费计算资源，没有必要。

查阅文档发现glfw本身已经提供了绘制点的方法：glDrawArrays(GL\_POINTS, i, 1)，因此直接调用本绘制点的方式，可以实现绘制出正方形的点，只需要完成两步：1. 将正方形的点变成圆形，2.实现不同的顶点的大小闪烁，而且大小不同。

对于1：在绘制顶点时的fragment shader文件当中，对绘制距离大于uniform变量distance的像素，统一进行discard处理，便可以将原本正方形的点变为圆形。

对于2：对每一个点设置不同的distance，并和改变颜色十分类似，利用cos函数的周期性完成光斑闪耀的功能即可。

（以下是开启高亮边和光斑闪烁功能后的画面效果）：



**⑤实现A键启动光斑传递（必须先S键开启高亮边和光斑闪耀功能）：**

这个功能比较复杂，为了简化这个问题，我将这个功能分为以下几步来实现：

1. 创建一个包括起点和终点、斜率、长度的line类，并获取三角形面片这个复杂图

当中的所有不重复边，并为其设定一个光斑顶点和一个光斑终点

2. 利用这个边的斜率k，来决定光斑在每一时刻的位移：当x位移了一个单位长度，

由点斜式方程：，y位移长度为k。

3. 设定一个光斑传递速度基本值speedBase，并对于每一条边上的光斑，将其传递速度

在基准值的基础上乘以这一条边的长度，这样可以保证所有的光斑都在同一时刻到

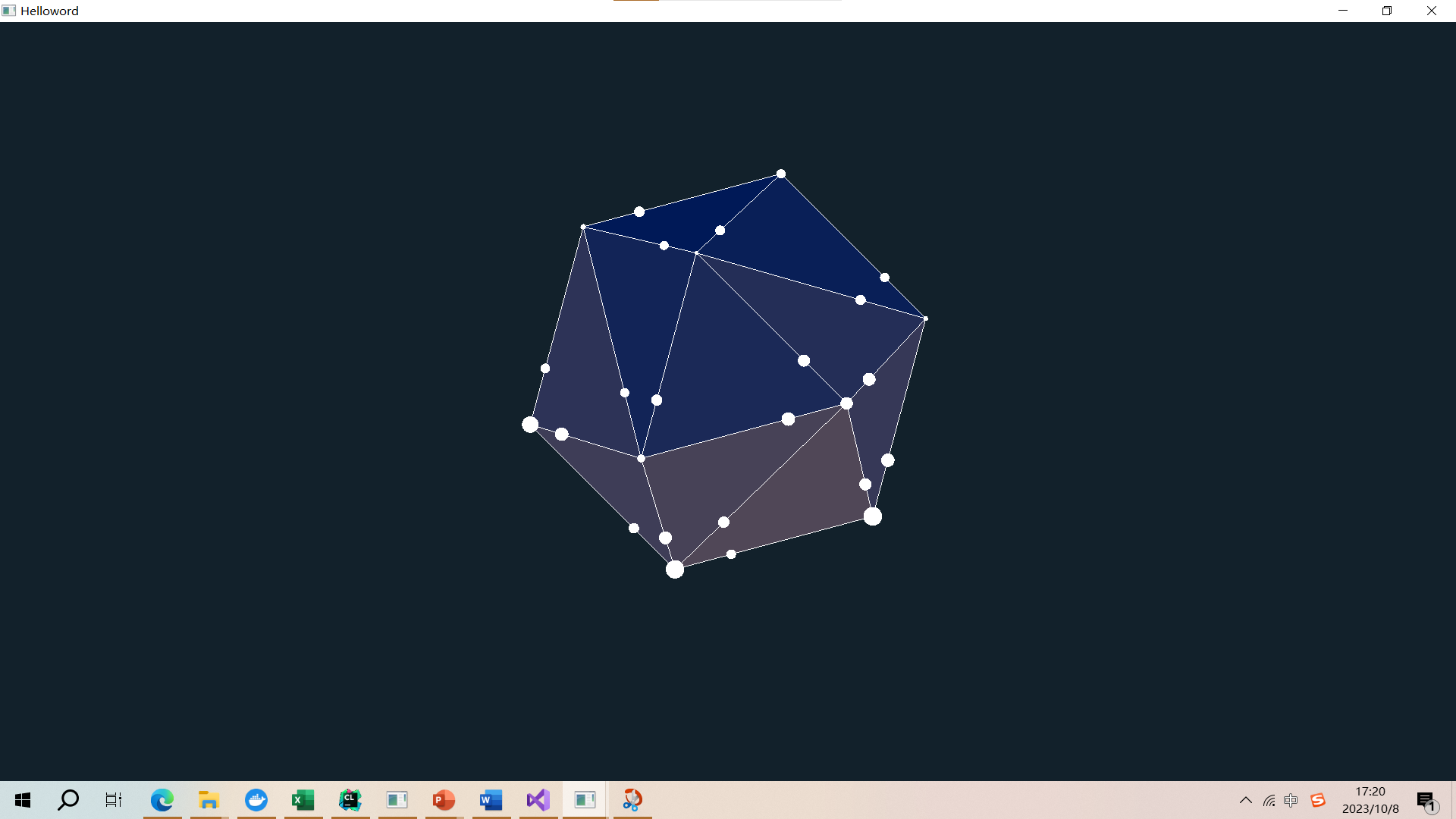
达终点，也会在同一时刻从起点出发，而没有必要为每一个光斑来保存位置和速度。

当有一个光斑到达终点的时候，光斑位置重置即可。

4. 和步骤④类似，也实现每个移动光斑以cos函数周期闪耀、大小不一的效果，并修改

个别参数使得其显示更加顺滑。

（开启光斑传递后的视觉效果：）

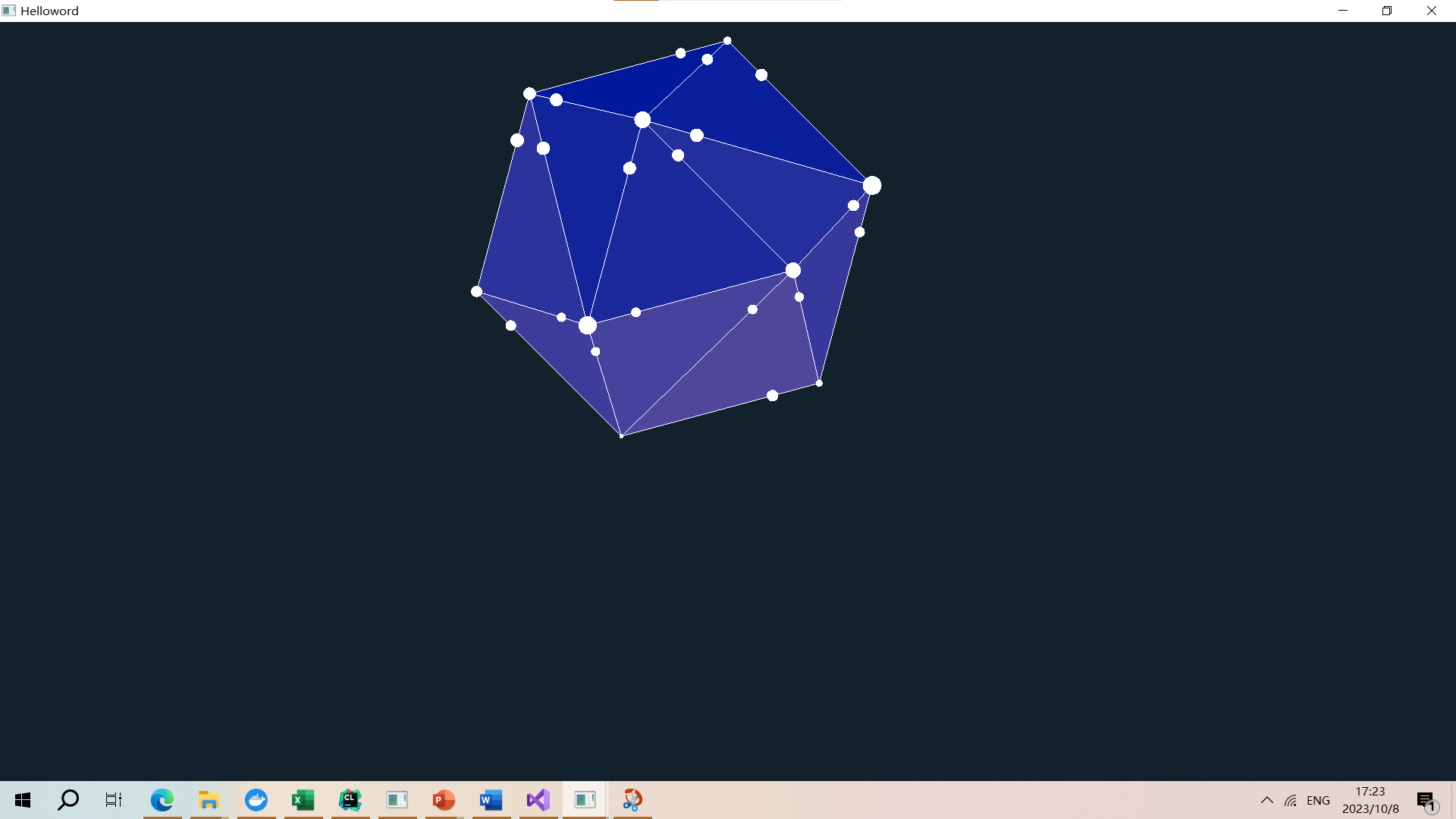


**⑥实现D键让图形做圆周运动的功能：**

这个功能实现比较容易，只要在point shader文件当中添加一个Uniform变量pointOffset，将每一个输入的二维点值进行更改：

这样就实现了让图形绕着原点做圆周运动的功能，这里对sin(timeValue)/3限定了它的移动范围：在一个距离屏幕中心半径为1/3屏幕长度的圆内。

（下图为移动效果，此时图形移动到了上方）



1. **程序使用说明**

点击程序源代码目录下的start.exe程序即可开启程序

**C键**：颜色渐变，再按取消

**S键**：显示高亮边以及顶点光斑，再按取消

**A键**：光斑传递（必须在显示高亮边和顶点光斑的情况下才会生效），再按取消

**D键**：光斑圆周运动，再按归位到原点

**ESC**：退出程序