CS30800
Introduction to Computer Graphics
Lab 4 – Quaternion and arcball

2025. 04. 01/ 2025. 04. 03

Note

- Note: we will not answer questions directly related to the assignment
  - The goal is to find solutions based on your understandings for the course.
- In the lab session, we will provide explanations about
  - Purpose of the assignment
  - What you need to implement
  - Recap for the background knowledge
- Some slides are from lecture notes of this course

Contents

- Quaternion
- Homework 3
- Q&A

Note

- Assignment 3 is built upon assignment 2

- Copy the following starter code into your project directory for asst2
  - arcball.h
  - quat.h
  - rigtform.h

Quaternion

Quaternion

- Quaternion is implemented in quat.h
- All operations are already implemented in provided code

- Why not using an Euler(x, y, z) rotation or a rotation matrix?
  - Euler rotation: Gimbal lock problem
  - Rotation matrix: too much computation

Quaternion

- Constructors

```
Quat() : q_(1,0,0,0) {}
Quat(const double w, const Cvec3& v) : q_(w, v[0], v[1], v[2]) {}
Quat(const double w, const double x, const double y, const double z) : q_(w, x,y,z) {}
```

  - For given axis and angle θ,

$$x = \sin\left(\frac{\theta}{2}\right)\cdot axis.x,\ y = \sin\left(\frac{\theta}{2}\right)\cdot axis.y,\ z = \sin\left(\frac{\theta}{2}\right)\cdot axis.z,\ w = \cos\left(\frac{\theta}{2}\right)$$

- Static constructors

```
static Quat makeXRotation(const double ang)
static Quat makeYRotation(const double ang)
static Quat makeZRotation(const double ang)
```

Quaternion

- Perform the following triple quaternion multiplication

$$\begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix} \begin{bmatrix} 0 \\ \hat{\mathbf{c}} \end{bmatrix} \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\hat{\mathbf{k}} \end{bmatrix}^{-1}$$

- Result is form of: $\begin{bmatrix} 0 \\ \hat{\mathbf{c}}' \end{bmatrix}$

- The vector multiplication is already implemented in skeleton code

```
Cvec4 operator * (const Cvec4& a) const {
  const Quat r = *this * (Quat(0, a[0], a[1], a[2]) * inv(*this));
  return Cvec4(r[1], r[2], r[3], a[3]);
}
```

Task 1: RigTForm class implementation

HW3 Goals: Task 1

- Implement Rigid Body Transformation class (RigTForm)
  - It consists of translation T (3D point vector) and rotation R (4D quaternion vector)
  - Efficient than matrix multiplication
  - It is a helpful utility class for further implementation

- Implement manipulations based on RigTForm class
  - Inversion
  - Multiplication
  - Conversion to a matrix
  - Conversion from a translation vector
  - Conversion from a quaternion
  - Multiplication with a vector

HW3 Goals: Task 1

- Alternate Matrix4 class with RigTForm class in the code before
  - You will use RigTForm class instead of Matrix4 class for manipulations

- After you replace Matrix4 by RigTForm, everything should behave sa me as before

HW3 Goals: Task 1 (Hint)

- RigTForm inversion
  - Rotation is inverse of the quaternion
  - Translation is affected by rotation of itself

$$\left(\begin{bmatrix} i & t \\ 0 & 1 \end{bmatrix}\begin{bmatrix} r & 0 \\ 0 & 1 \end{bmatrix}\right)^{-1} = \begin{bmatrix} i & -r^{-1}t \\ 0 & 1 \end{bmatrix}\begin{bmatrix} r^{-1} & 0 \\ 0 & 1 \end{bmatrix}$$

- RigTForm multiplication
  - Rotation is multiplication of two quaternions
  - Translation of second RigTForm is affected by rotation of first RigTForm
  - (Translation is always affected by previous rotation)

$$\begin{bmatrix} i & t_1 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} r_1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} i & t_2 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} r_2 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} i & t_1 + r_1 t_2 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} r_1 r_2 & 0 \\ 0 & 1 \end{bmatrix}$$

Task 2: Arcball

HW3 Goals: Task 2

- Implement the arcball interface
  - Draw a sphere to represent the arcball
  - Implement an arcball function in OpenGL functions

- Compute rotation
  - Compute two 3D vectors on the screen space

- Two helper functions in skeleton code
  - getScreenSpaceCoord
  - getScreenToEyeScale

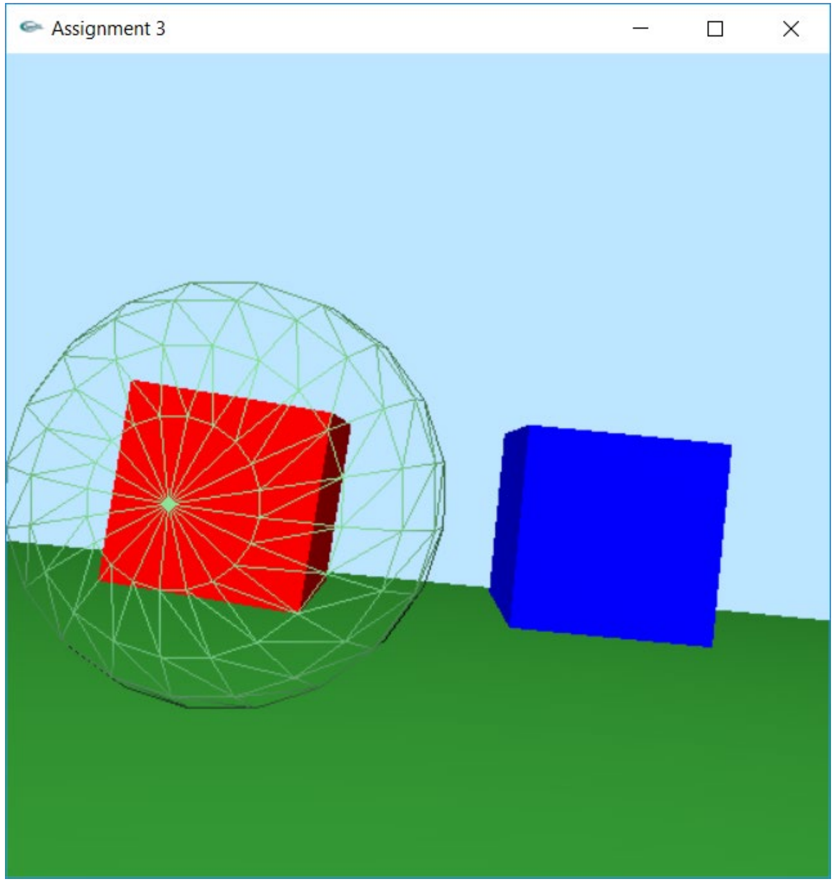
- The radius of the sphere should be
  - 0.25 * min(g_windowWidth, g_windowHeight)

HW3 Goals: Task 2

- Draw a wire framed sphere for arcball visualization

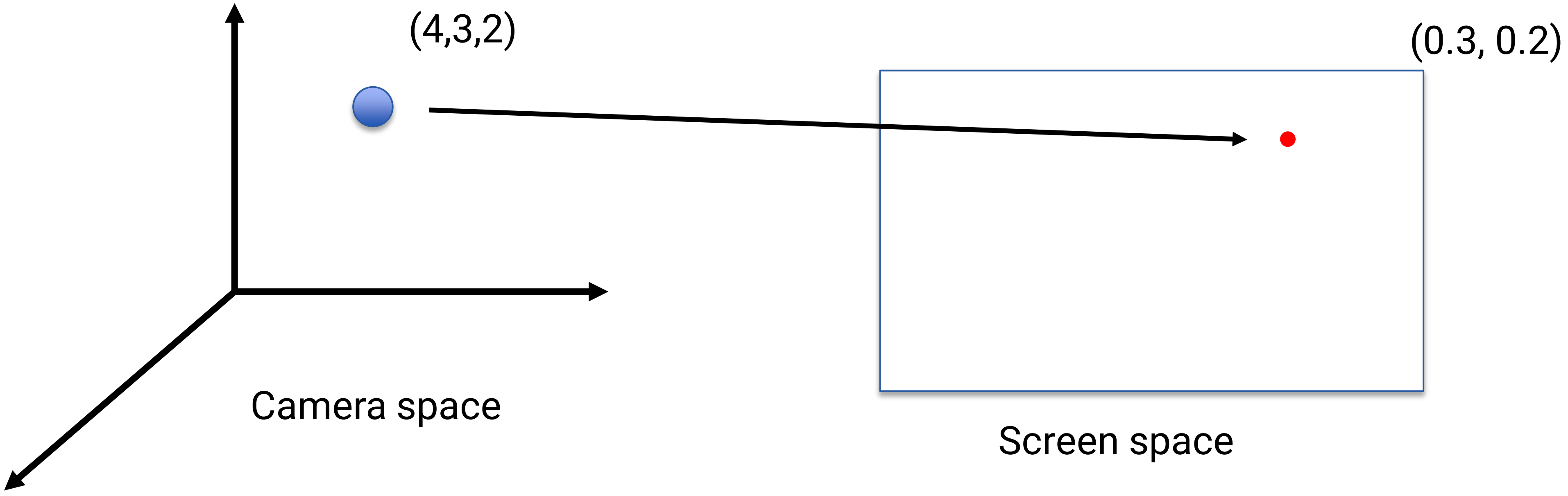
```
// switch to wire frame mode
glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);

// draw something
g_sphere->draw(curSS);


// switch back to solid mode
glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
```

(If you do not switch back to “GL_FILL”, then all objects may be drawn in wireframe)



HW3 Goals: Task 2

- getScreenSpaceCoord
- Convert a 3D vector of the point to a 2D point on a screen space (in pixel units)



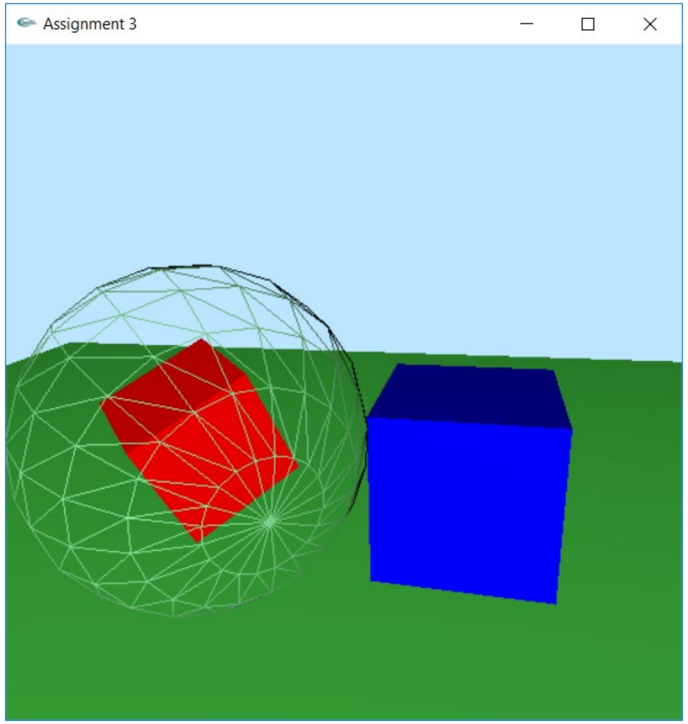

```
inline Cvec2 getScreenSpaceCoord(const Cvec3& p,
                                 const Matrix4& projection,
                                 double frustNear, double frustFovY,
                                 int screenWidth, int screenHeight)
```

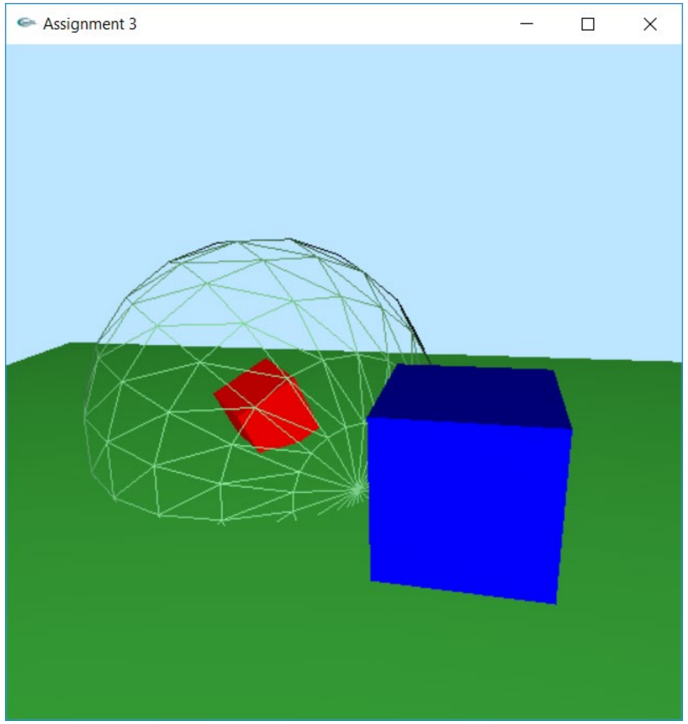
HW3 Goals: Task 2

- getScreenSpaceCoord
- Why should we use it?
  - Mouse position is given as 2D point on a screen space
  - Arcball is a 3D object.
  - We should convert one of two points (mouse position, arcball position) to the other coordinate to calcu late arcball manipulation

HW3 Goals: Task 2

- getScreenToEyeScale
- Arcball size should not be changed in Screen Space even if size of the cube is ch anged in screen space due to translation

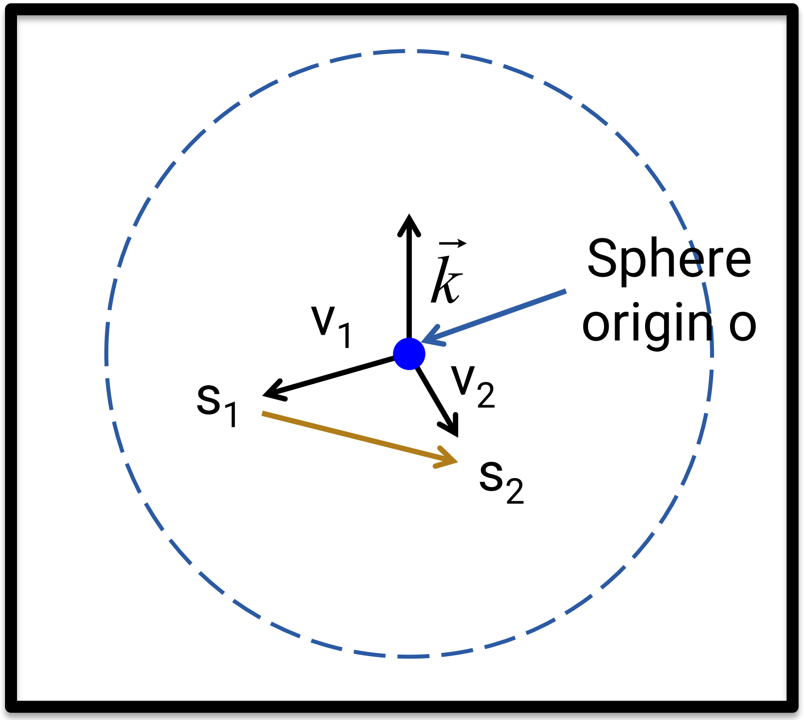


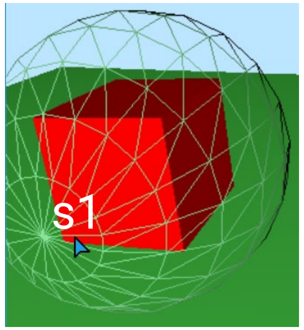


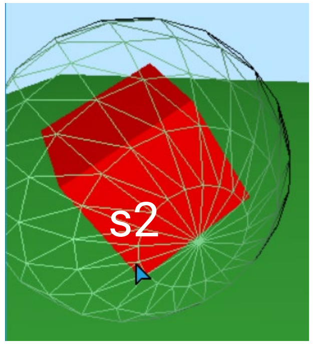

```
inline double getScreenToEyeScale(double z, double frustFovY, int screenHeight)
```

HW3 Goals: Task 2

- Sphere origin o: center of sphere, projection of a frame origin
- S_1: clicked screen coordinate
- S_2: dragged mouse screen coordinate

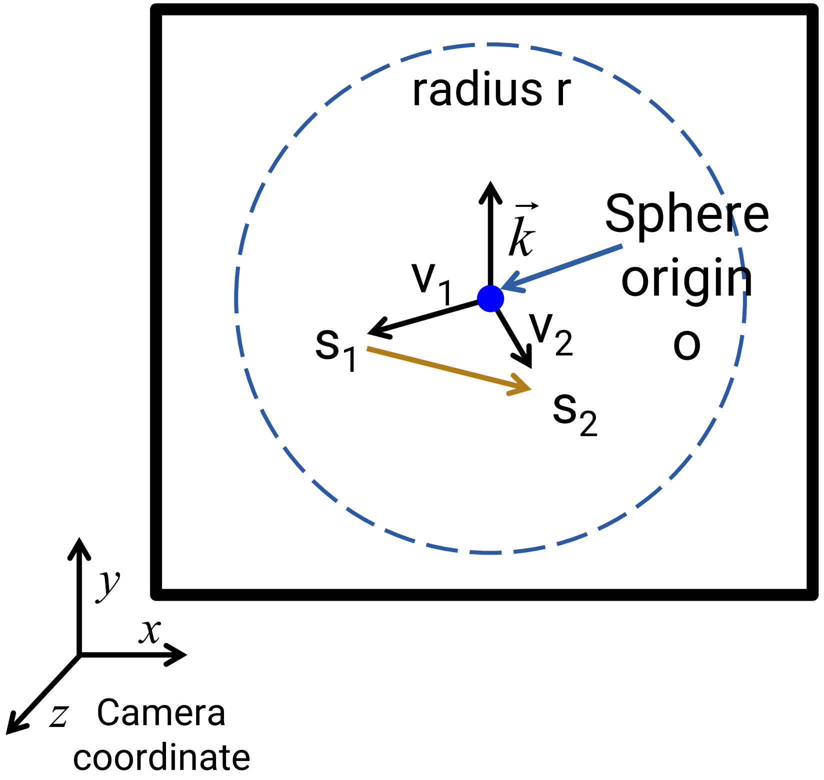






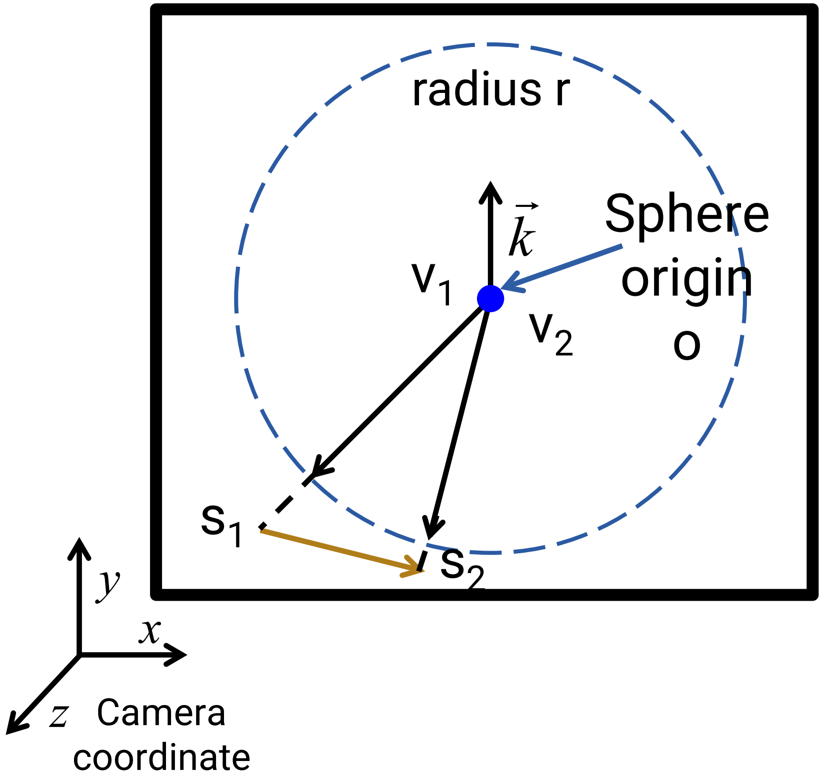
HW3 Goals: Task2 (Hint)

- v_1, v_2 – the directional vectors
- $v_{1x} = s_{1x} - o_x, v_{y1} = s_{1y} - o_y, v_{1x}^2 + v_{1y}^2 + v_{1z}^2 = r^2$



HW3 Goals: Task2 (Hint)

- When you drag outside of the arcball, use nearest point of the arcball for manipulation



Task 3: Translation fix up

HW3 Goals: Task 3

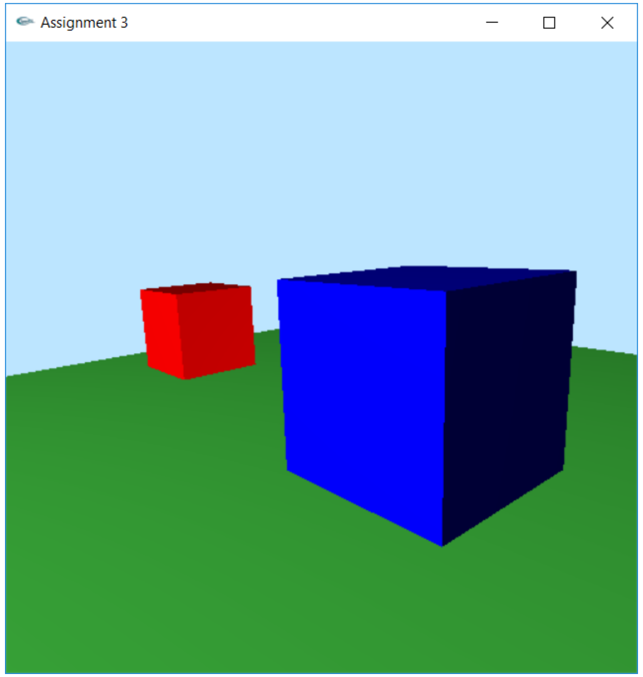
- Translation fix up
- Translate the object as same as the mouse movement

- Use g_arcballScale

- Wherever the object is, the object should follow a mouse pointer

HW3 Goals: Task 3

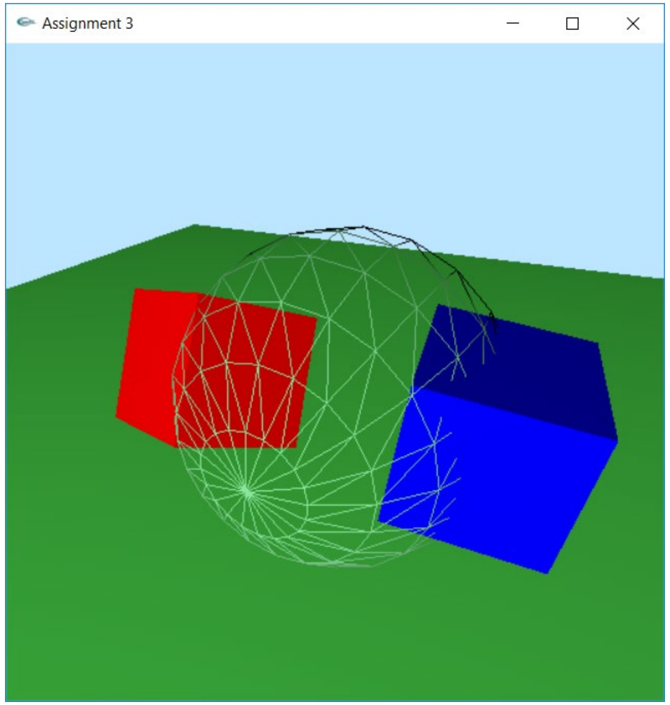
- For same size of objects, far objects look smaller
- For same translation, far objects look translate smaller than near objects
- **However, translation of HW3 should be same as translation of mouse cursor**
- **You can scale the translation based on the g_arcballScale variable**
- We will learn perspective projection later



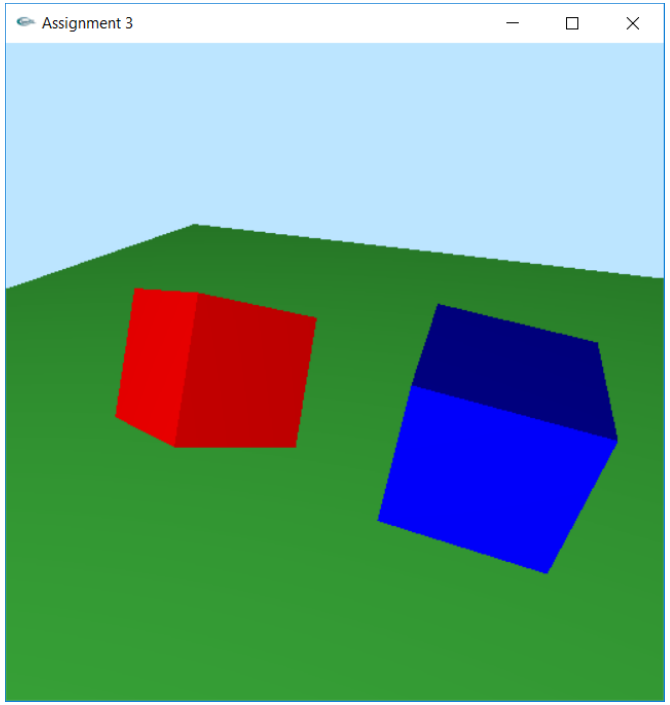
HW3 Goals: Task 3

- When ‘m’ is pressed

World-sky frame



sky-sky frame (Ego-motion)



HW3 Goals: Task 3

- Note that
- There is a statement in document:
  - When the arcball is not in use (e.g., in ego motion), g_arcballScale may not be correctly defined, so feel free to fall b ack to the hard-coded number in that case.

- Which means translation should not be scaled in sky-sky mode.

Question?