Camera path

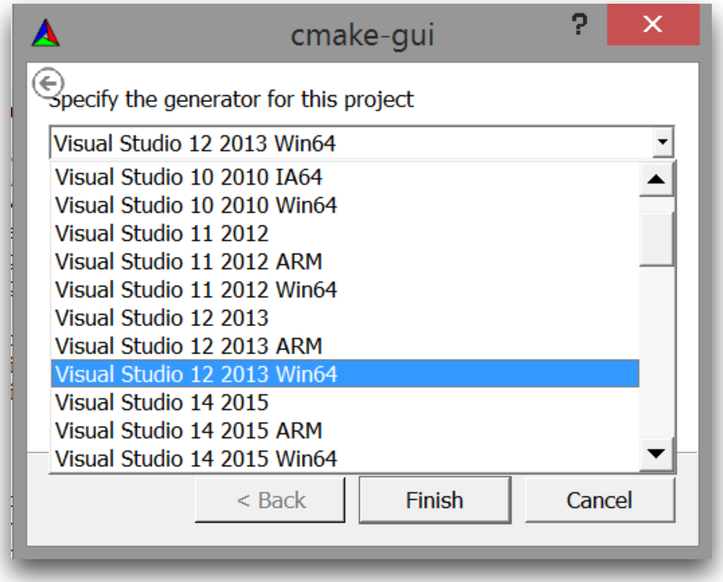
Minh Dang

Structure of Practicals

Week	Topic	Practical	Homework
2	2D OpenGL	Intro to GLSL Interfacing GLSL / C++ Working with Textures	Triangle Spirals Checkerboard 2D Planet System
3	3D OpenGL	3D Transformations Orthographic Projections Index Buffers	Perspective Projections Virtual Trackball Animated Triangle Grid
4	Screen Space Techniques	**invited talk:** R.Ziegler (45m) Anttweakbar: intro Gouraud shading	Shading: phong, flat, toon, artistic Spot light
5	Parametric Curves	Picking Bezier evaluation	Piece-wise cubic bezier Screen-space unprojection Camera paths

Preliminaries

- Code in the git repository
  - `git clone https://git.epfl.ch/repo/icg15.git`
  - `git pull` (to update already checkout code)
  - if `pull` doesn't work: `git reset –hard HEAD`
    ATTENTION: Backup before doing this
- For Windows users: Use x64 compiler
- These slides are in `PDF/Practical-Homework#5.pdf`

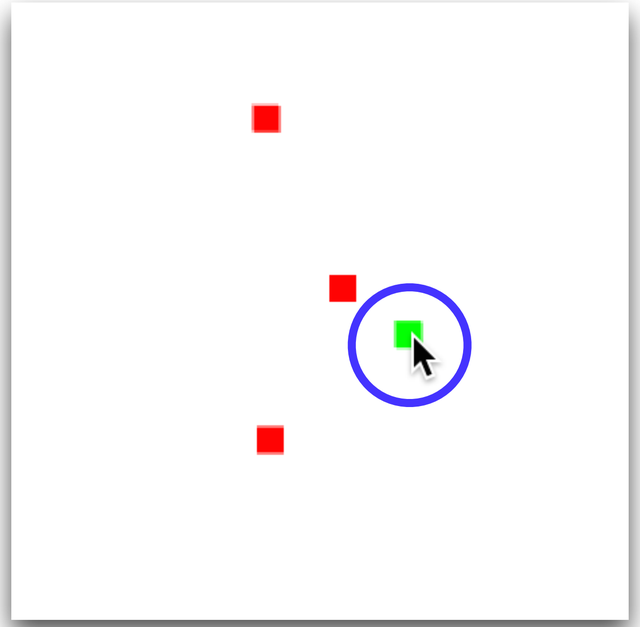


Practical 5

- picking
- bezier
- code in `lab5_bezier`

Picking

- Select an object by mouse
- Useful tutorial: http://www.lighthouse3d.com/tutorials/opengl-short-tutorials/opengl-selection-tutorial/
- Steps
  1. Set different id for each object
  2. Render an object with color based on its id
  3. ***glReadPixels(…)*** to get the pixel color under mouse position, thus object id



Picking

Picking objects: `ControlPoint (_point/point.h)`

Step 1: ids are set in `main.cpp`

```
for (unsigned int i = 0; i < cam_pos_points.size(); i++) {
    cam_pos_points[i].id() = i;
    ...
}
```

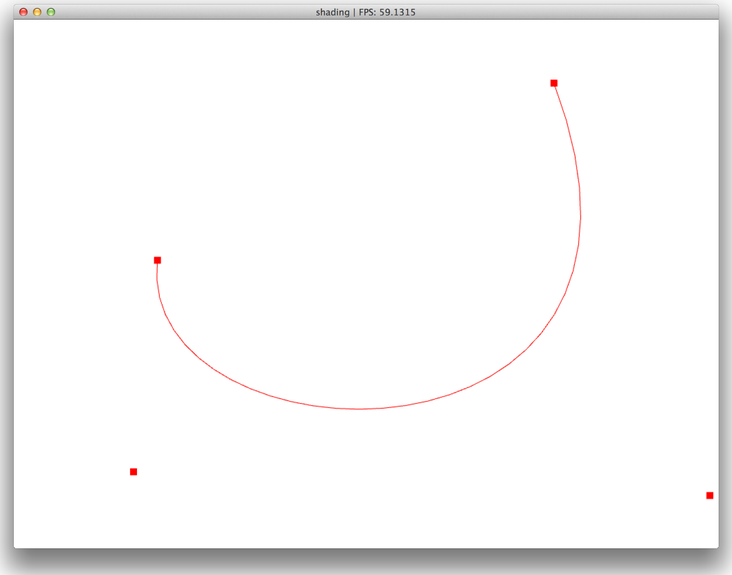
Step 2:

- **TODO P5.1**: complete `_point/point_selection_fshader.glsl`
- **TODO P5.2**: complete `render_selection() (main.cpp)`

Step 3: Look at `void selection_button(int button, int action){...}`

Bezier curve

- Parametric curves widely used in computer graphics
- We will use these curves to control the camera (Homework)
- Base code: _bezier/bezier.h
- **TODO P5.3** complete `void bezier(Hull& p, int depth=0)`
  - Use “de Casteljau” subdivision algorithm to evaluate the curve
  - 5 splits
  - Split at \alpha = 0.5
  - Save the points to _vertices
- Move the points around to see how the curve changes



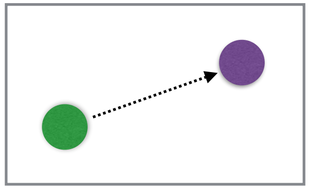
Editing the control points

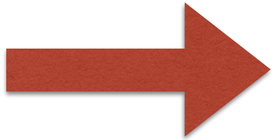
- Use the mouse to move the selected point
- Need to convert from mouse position into world position
  - Look at `bool unproject (int win_x, int win_y, vec3 &p)` in main.cpp
  - This only works for orthographic camera defined in this practical.

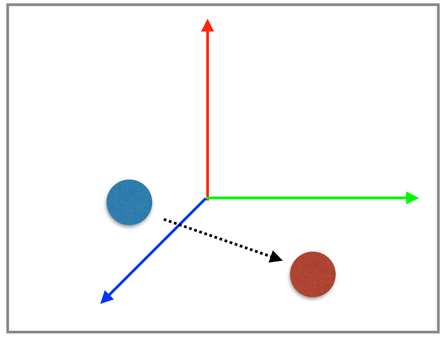
Homework 5

- screen-space unprojection **2 points**
- piece-wise cubic bezier **1 points**
- camera path **3 points**
- deadline: **Thursday 26/03, 10 am**
- code in `hw5_bezier`
- office hours: 1pm - 2pm Monday, Tuesday, Wednesday
- note: replace the practical TODOs (TODO P5.x) by the practical solution

1. Screen-space unprojection



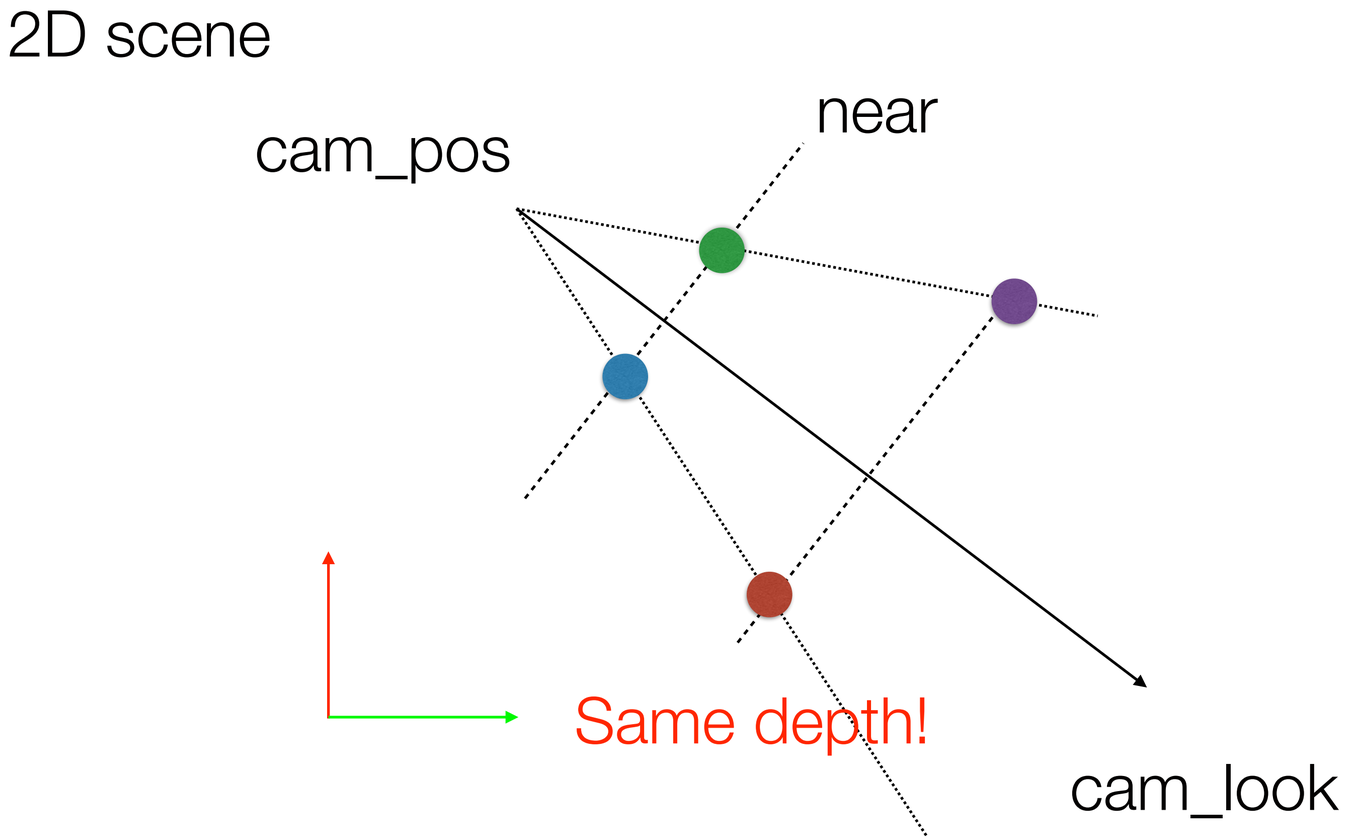




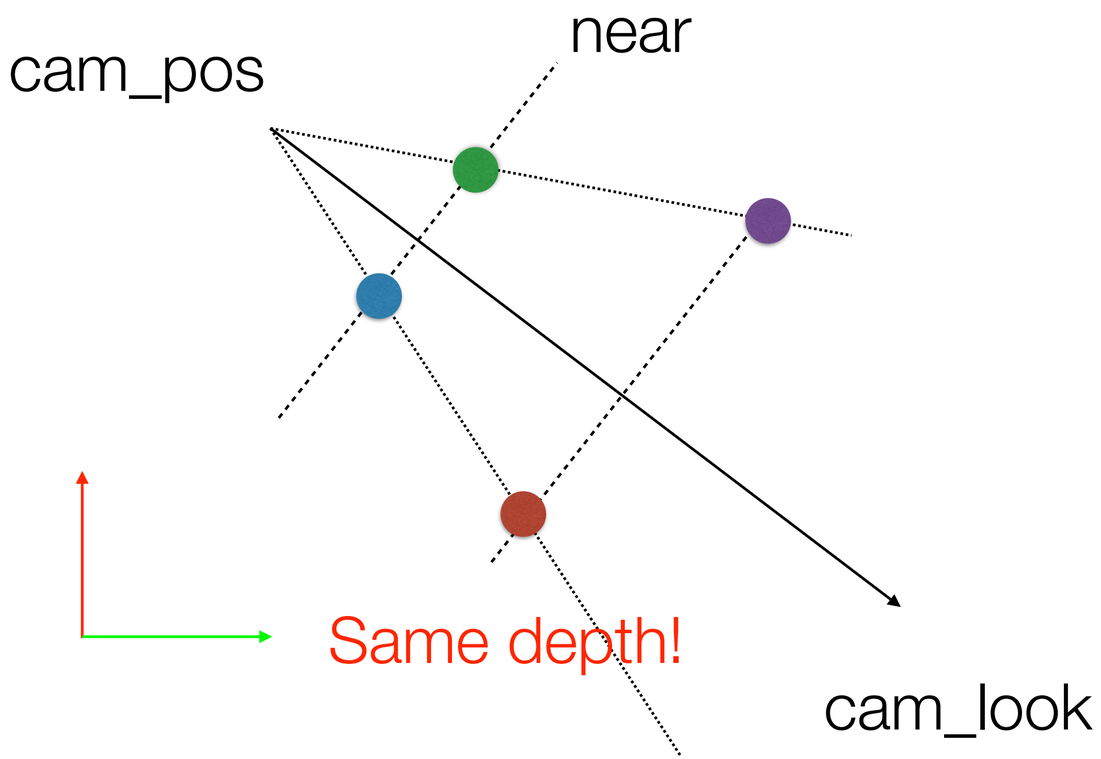
mouse movement
(in viewport)

object movement

1. Screen-space unprojection



1. Screen-space unprojection



How to find the new point?

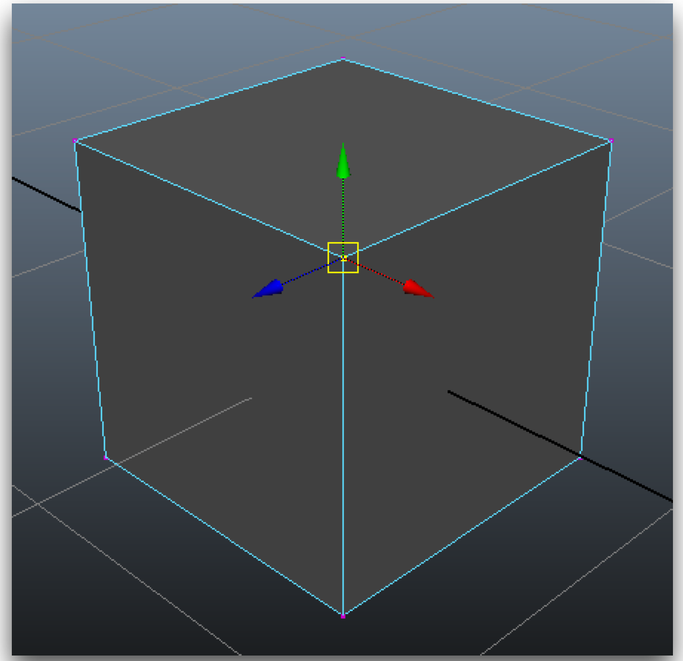
- p_screen = MVP * p
- Step 1: find p_screen from the mouse position
- Step 2: p = inv(MVP) * p_screen
- Convert homogeneous coordinates to Cartesian coordinates (divide by the fourth component)

TODO H5.1 complete `unprojection(), main.cpp`

We prepare the Trackball for you. Use left Shift modifier to rotate the scene.

Bonus. Translation Gizmo

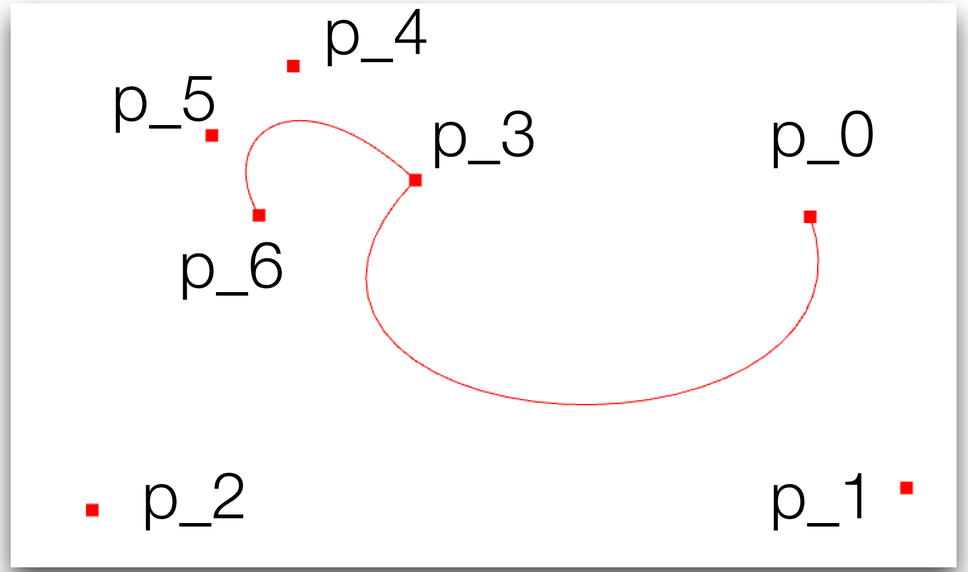
- Also called manipulators or widgets
- Use to visually translate an object
- Task: implement a gizmo to move control points
  - Draw an axis aligned gizmo for each selected points
  - Move the point in x/y/z directions by dragging the red/green/blue axis of the gizmo
- **1 points**



2. Piece-wise cubic bezier

- Combine multiple cubic bezier segments
  - Control points: p_0, p_1, … p_6
  - First segment: p_0, p_1, p_2, **p_3**
  - Second segment: **p_3**, p_4, p_5, p_6

- **TODO H5.2** Modify bezier.h so that it can handle piece-wise bezier curve. You may have to change some function signatures. Update main.cpp accordingly.

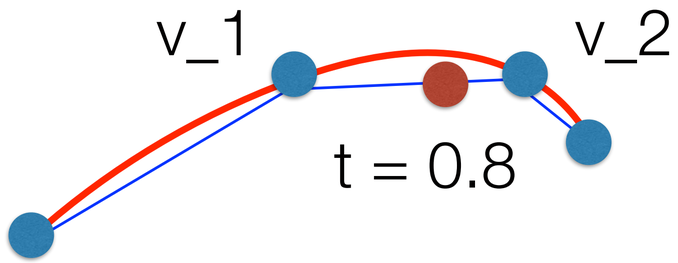


3. Camera path

- Camera specifications
  - cam_pos
  - cam_look
  - cam_up
  - `mat4 view = Eigen::lookAt(cam_pos, cam_look, cam_up);`
- Animate the camera
  - sample cam_pos_curve to get cam_pos
  - sample cam_look_curve to get cam_look
  - the view matrix changes over time

3. Camera path

- arc-length parametrization
  - bezier curves do not possess a natural arc-length parametrization
  - velocity of camera along the path is not constant
- simple solution to uniform sample the curve
  - total_length: total length of the curve
  - get t in [0, 1]
  - sample point at distance t * total_length from start point
  - find two closest vertices v_1 and v_2
  - find a point in the line connecting v_1 and v_2



3. Camera path

TODO H5.3 uniformly sample a bezier curve

- complete `void compute_parameterization()`
  to calculate the curve distance from start point to the vertex i **(1 points)**
- complete `void sample_point(double t, vec3 &sample)`
  to sample a point `sample` according to `t` in `[0, 1]` **(1 points)**
- set the control points for `cam_look_curve` in `init()` of `main.cpp` (0.5 point)
- complete the `display()` to sample `cam_pos` and `cam_look` from corresponding curves. **(0.5 point)**
- your curves must have at least two segments and editable!

Bonus. Load a nice mesh, shade it using HW4, and design a nice camera path to explore the mesh. **(0.5 point)**