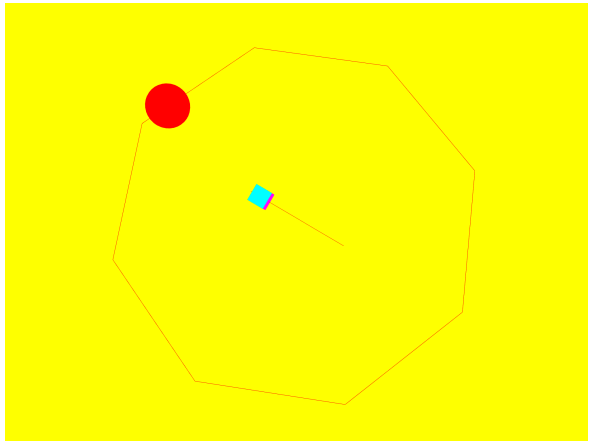
UNIVERSITÉ Concordia UNIVERSITY

COMP 371
Computer Graphics

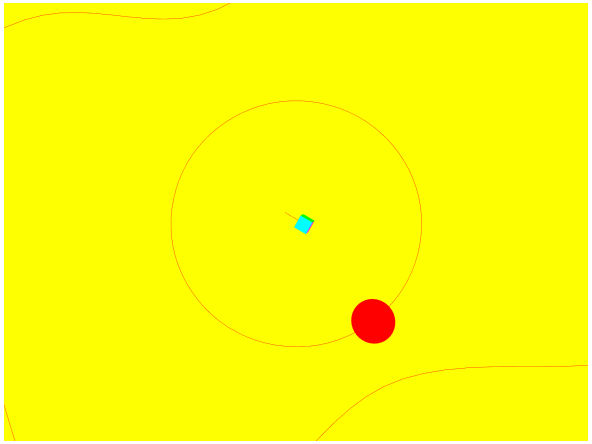
Lab 05
Uniform B-Spline interpolation

Prepared by Nicolas Bergeron

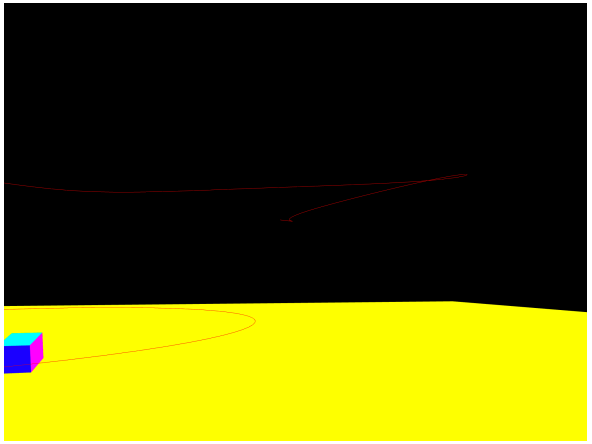
Expected results



A1



Spline Interpolation
for positions



cinematic Camera
(on a spline)

Getting Started

- This will improve the position interpolation of A1.
- Positions will move on a smooth cubic curve (Uniform B-Spline) instead of straight line.
- It will add a cinematic camera in your framework
- Add the provided files to the framework (this may overwrite the particle system bits)
  - If you are good with git, use a feature branch to make the integration easy with the particle systems
  - Otherwise, just start from your solution of A1

Outline for Lab05
Uniform B-Spline interpolation

- Improve position interpolation for Assignment 1
  - Current interpolation (LERP) is between 2 points
  - This version involves 4 points to create the curve

- Overwrite your A1 files with the ones provided, don't forget to add the new scene file.

- Add a cinematic camera to the framework (moving on B-Splines, through key frames)

TUTORIAL
BSPLINE INTERPOLATION

B-Spline Interpolation

The equation below describes the curve segment generated by a Uniform Cubic B-Spline with control points P_i, P_{i+1}, P_{i+2} and P_{i+3} at parametric time t.

- $$BSpline(t) = \frac{1}{6}\begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}\begin{bmatrix} P_i \\ P_{i+1} \\ P_{i+2} \\ P_{i+3} \end{bmatrix} \quad \text{for} \quad t \in [0,1]$$

The derivative of the function will give a tangent vector on the spline. You will implement this later…

- $$BSpline'(t) = \frac{1}{6}\begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix}\begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}\begin{bmatrix} P_i \\ P_{i+1} \\ P_{i+2} \\ P_{i+3} \end{bmatrix} \quad \text{for} \quad t \in [0,1]$$

Integrate Splines for Animation position interpolation (1/3)

- Add Spline to Animation class, in Animation.h

```
#include "BSpline.h"
```

```
std::vector<AnimationKey> mKey;
std::vector<float> mKeyTime;

BSpline spline;
```

Integrate Splines for Animation position interpolation (2/3)

- In Animation.cpp, update Drawing Code
- Add each position except last to avoid duplicates

```
void Animation::CreateVertexBuffer()
{
    // Instead of drawing lines between keyframe's position,
    // let's just add our positions to our spline and draw the spline
    for (int i=0; i<mKey.size()-1; ++i)
    {
        spline.AddControlPoint(mKey[i].GetPosition());
    }

    spline.CreateVertexBuffer();
}

void Animation::Draw()
{
    spline.Draw();
}
```

Integrate Splines for Animation position interpolation (3/3)

Spline position interpolation

The value of t for the spline is the integer of your first key + your normalized time between keys

Compare this code with your A1 solution

```
glm::mat4 Animation::GetAnimationWorldMatrix() const
{
    int key1 = 0, key2 = 0;
    float normalizedTime = 0.0f;

    for (int i=1; i<(int) mKey.size(); ++i)
    {
        if (mCurrentTime < mKeyTime[i])
        {
            key1 = i - 1;
            key2 = i;
            normalizedTime = (mCurrentTime - mKeyTime[key1]) / (mKeyTime[key2] - mKeyTime[key1]);
            break;
        }
    }

    // Translation Component: Uniform B-Spline interpolation
    // Provide the normalized time + the integer matching first control point to use
    vec3 position =  spline.GetPosition(key1 + normalizedTime);

    // Scaling Component: Linear Interpolation of Scaling Coefficients (LERP)
    vec3 scaling = glm::mix(mKey[key1].mScaling, mKey[key2].mScaling, normalizedTime);

    // Rotation Component : Spherical Linear Interpolation (SLERP)
    // Two rotation states for two keys to interpolate from
    // Spherical Linear Interpolation
    quat rotation1 = angleAxis(radians(mKey[key1].mRotationAngleInDegrees), mKey[key1].mRotationAxis);
    quat rotation2 = angleAxis(radians(mKey[key2].mRotationAngleInDegrees), mKey[key2].mRotationAxis);
    quat rotation = glm::slerp(rotation1, rotation2, normalizedTime);

    // Concatenate interpolated components to a single world transform
    mat4 t = glm::translate(mat4(1.0f), position);
    mat4 r = glm::mat4_cast(rotation);
    mat4 s = glm::scale(mat4(1.0f), scaling);

    return t * r * s;
}
```

Check how curved lines are drawn in BSpline class – No code to write!

- It samples 10 points per line segment, before setting up the VAO and VBO (not really curves)

```
const int numPointsPerSegment = 10;
float increment = 1.0f / numPointsPerSegment;

for (int i=0; i < mControlPoints.size(); ++i)
{
    float t = 0.0f;

    // Set current control points
    vec3 p1 = mControlPoints[i];
    vec3 p2 = mControlPoints[(i + 1) % mControlPoints.size()];
    vec3 p3 = mControlPoints[(i + 2) % mControlPoints.size()];
    vec3 p4 = mControlPoints[(i + 3) % mControlPoints.size()];

    for (int j=0; j < numPointsPerSegment; ++j)
    {
        mSamplePoints.push_back(GetPosition(t, p1, p2, p3, p4));
        t += increment;
    }
}
```

Camera on Spline
Tangent vector for lookAt

- Implement GetTangent() functions, start from GetPosition() equivalents and replace with code in red

```
glm::vec3 BSpline::GetTangent(float t) const
{
    int p1 = ((int) t) % mControlPoints.size();
    int p2 = (p1 + 1)  % mControlPoints.size();
    int p3 = (p2 + 1)  % mControlPoints.size();
    int p4 = (p3 + 1)  % mControlPoints.size();

    return vec3(GetWorldMatrix() * vec4(BSpline::GetTangent(t - (int) t,
                                                            mControlPoints[p1],
                                                            mControlPoints[p2],
                                                            mControlPoints[p3],
                                                            mControlPoints[p4]), 0.0f));
}

glm::vec3 BSpline::GetTangent(float t, const vec3& p1, const vec3& p2, const vec3& p3, const vec3& p4)
{
    // Returns the position of the object on the spline
    // based on parameter t and the four control points p1, p2, p3, p4
    vec4 params(3*t*t, 2*t, 1, 0);
    mat4 coefficients(vec4(-1, 3, -3, 1), vec4(3, -6, 0, 4), vec4(-3, 3, 3, 1), vec4(1, 0, 0, 0));
    vec4 product = (1.0f / 6.0f) * params * coefficients;

    return vec3( vec4(product.x * p1 + product.y * p2 + product.z * p3 + product.w * p4, 1.0f) );
}
```

Camera on Spline
LookAt vector

- In main.cpp, load the BSplineScene.scene provided
- Press [3] to switch to spline camera.
- In BSplineCamera.cpp, add the following code:

```
void BSplineCamera::Update(float dt)
{
    // @TODO - Using the BSpline class, update the position on the spline
    //         Set the mLookAt vector pointing in front (tangent on spline)
    //         And mUp vector being as up as possible
    mLookAt = mpSpline->GetTangent(mSplineParameterT);

    // This is world units per variation of parameter t
    float tangentMagnitude = length(mLookAt);
    mLookAt = normalize(mLookAt);

    mSplineParameterT += mSpeed * dt / tangentMagnitude;
    mPosition = mpSpline->GetPosition(mSplineParameterT);

}

glm::mat4 BSplineCamera::GetViewMatrix() const
{
    return glm::lookAt(mPosition, mPosition + mLookAt, mUp);
}
```