Computer Graphics HW#3

공과대학 컴퓨터공학부

2020-19422 권신영

- Specifications

1. Build your own virtual roller coaster using splines

   B-spline으로 closed-spline을 만들고, 그 경로를 roller-coaster rail로 사용하였음

   (임의의 점을 추출하여 곡선 구현에 사용)

```
# Control points : x, y, height
ctrl_pts = [
    [0, 0, 0],
    [11, 3, 4],
    [8, 12, 8],
    [1, 8, 3],
    [-3, 12, 3],
    [-5, 8, 3],
    [-1, 7, 2]
]
```

   main.py – line 30

```
# arc-length parameterization과, 레일의 타일을 만드는 함수
def generate_rail_tile(self):

    ctrl = np.vstack([self.ctrl_pts, self.ctrl_pts[:self.degree]])
    n = len(ctrl)

    # B-spline을 사용함 -> knot vector 정의
    knot = np.arange(0, n + self.degree + 1)

    t_dense = np.linspace(self.degree, n, 10 *self.samples, endpoint=False)
    spline = BSpline(knot, ctrl, self.degree)
    pts_dense = spline(t_dense)
```

   Primitives_hw3.py - line 102

   Self.ctrl_pts[:self.degree]를 통해 closed-B-spline을 구현하였음

2. Define the moving coordinate frame along the spline.

```
# frenet frame용 원통 3개
self.tangent_cyl = Cylinder(radius=0.5, height=2, colors=[0,0,255,255])
self.normal_cyl = Cylinder(radius=0.5, height=2, colors=[255,255,0,255])
self.binormal_cyl = Cylinder(radius=0.5, height=2, colors=[255,0,0,255])
```

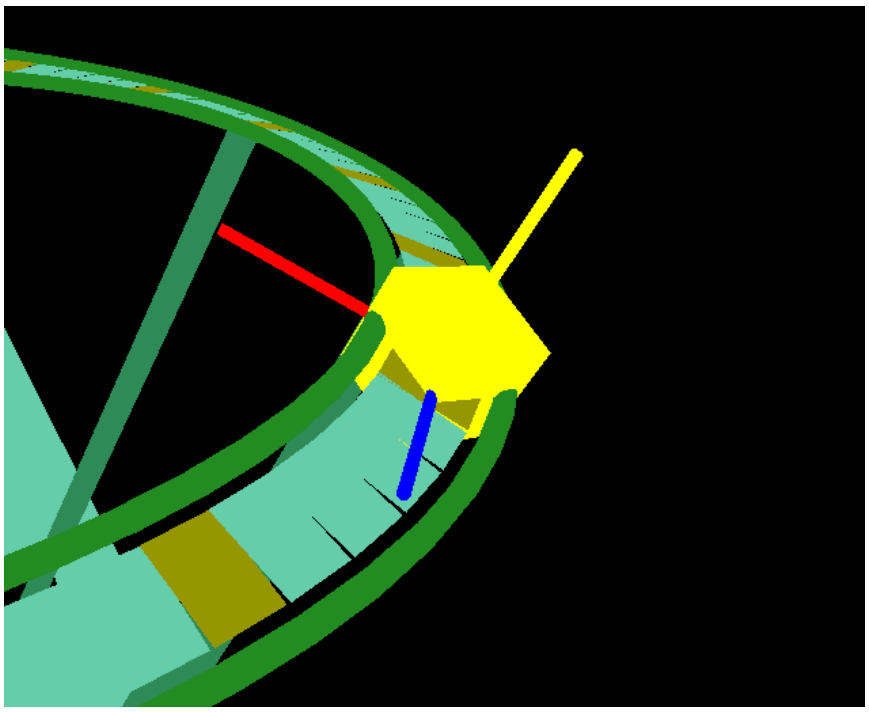
   Render_hw3.py – line 60

```
# Frenet Frame 계산
tangent = p2 - p1
tangent /= np.linalg.norm(tangent) + 0.000000001
```

```
up = np.array([0, 1, 0])
if abs(np.dot(tangent, up)) > 0.99: up = np.array([1, 0, 0])

# gram-schmidt
proj = np.dot(up, tangent) * tangent
normal = up - proj
normal /= np.linalg.norm(normal) + 0.000000001

binormal = np.cross(tangent, normal)
binormal /= np.linalg.norm(binormal) + 0.000000001

self.tangent_shape.transform_mat = self.get_transform(tangent, cart_pos)
self.normal_shape.transform_mat = self.get_transform(normal, cart_pos)
self.binormal_shape.transform_mat = self.get_transform(binormal, cart_pos)
```

Render_hw3.py – line 164~186

Gram-schmidt를 적용하여, tangent를 제외한 두 축이 급격이 변화하는 것을 방지하였음.



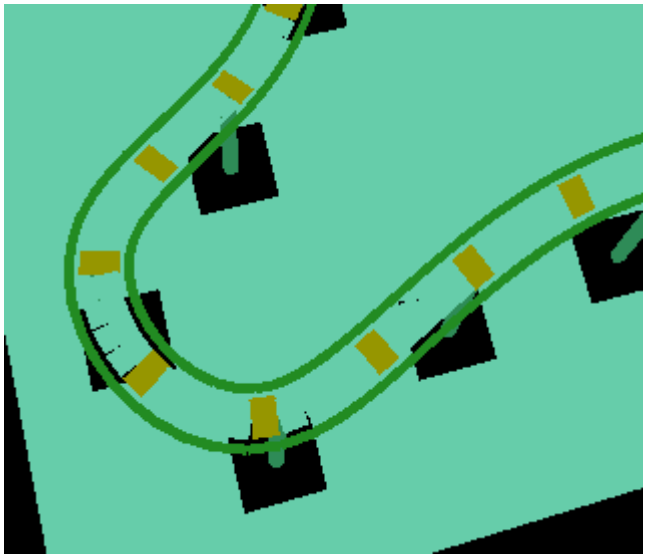
Trackball viewer를 통해 본 모습) Frenet Frame을 시각화하였음. 파란색이 Tangent, 노란색이 Normal, 빨간색이 BiNormal 축임. (tan X Nor = BiNor 확인 가능)

3. Simulate the coaster to create physically correct motion along the spline.

```
s_target = np.linspace(0, total_length, self.samples)
reparameterized_t_vals = []

index_pointer = 0
for s_target in s_target:
    while index_pointer < len(s_lst)-1 and s_lst[index_pointer+1] < s_target: index_pointer += 1

    s0, s1 = s_lst[index_pointer], s_lst[index_pointer+1]
    t0, t1 = t_dense[index_pointer], t_dense[index_pointer+1]

    # 핵심) t 선형 보간
    alpha = (s_target - s0) / (s1 - s0 + 0.00000001)
    t_interpolation = (1 - alpha) *t0 + alpha * t1

    reparameterized_t_vals.append(t_interpolation)

points = spline(reparameterized_t_vals)
self.path_points = points

# arc-length parameterization ends
```

primitives_hw3.py – generate_rail_tile function

이 arc-length parameterization과, primitives_hw3.py의 find_t_from_s function을 통해 곡선 상에서의 균일한 거리 점들을 추출해내었음 -> 이를 통해 속도를 구현



이를 이용하여 트랙의 일정한 거리마다 기준이 되는 타일을 구현) arc-length parameterization이 없었다면, 이 타일의 거리는 많이 휘어지는 부분에서 더 좁은 간격을 보일 것(실제로 적용하지 않았을 때 이 현상이 일어났음)

```
# arc-length 기반..
self.cart_s += self.cart_speed * dt
if self.cart_s >= self.rail.total_length: self.cart_s -= self.rail.total_length

t = self.rail.find_t_from_s(self.cart_s)
point_1 = np.array(self.rail.spline(t))
y = point_1[1]

g = 9.8
y_max = np.max([pt[1] for pt in self.rail.path_points])  # 높이 최댓값

# 최고점에서 최소 속도 세팅
v = math.sqrt(2 *g *max(0.0, y_max -y)) + 0.01

self.cart_s += v * dt
```

Render_hw3.py – line 131

에너지 보존 법칙을 통한 속도 구현 + 최고점에서의 최소 속도 보정(0.01)

실제로는 보정을 넣지 않아도 아주 작은 속도가 존재하기는 함.

```
# 작은 큐브: 움직이는 '카트'역할을 합니다.
cart = Cube(scale_x=0.5, scale_y=0.5, scale_z=0.5, colors=[255,255,0,255])

start_pos = Vec3(*rail.path_points[0])
cart_transform = Mat4.from_translation(start_pos)
renderer.cart_shape = renderer.add_shape(
    transform=cart_transform,
    vertice=cart.vertices,
    indice=cart.indices,
    color=cart.colors,
    type="cart",
    group=None
)
```

Main.py – line 47

카트는 노란색 큐브로 인해 구현, 다만 Frenet frame의 tangent가 접선 방향을 명확히 구현하므로, 카트의 회전까지 구현하지는 않았음.

4. Render the simulating in a first-person view.

이후 조작 방법에서 설명.

- **조작법**

  Space를 누르면 카트가 움직이기 시작함. 이 때, 초기 camera 상태는 trackball viewer 가 적용된 상태로, 전체 트랙의 모습과 카트의 움직임을 잘 확인할 수 있도록 하였음. (이 상태에서 shift, alt 등의 trackball viewer 조작 가능)

  Space로 카트를 멈출 수 있음.

  First Person View: 최초 움직임이 시작된 이후, F키를 눌러 1인칭 시점으로 변환할 수 있음. 다시 F키를 눌러 바로 이전의 trackball viewer 시점으로 돌아올 수 있다.

  왼쪽이 최초 상태, 오른쪽이 space를 눌러 움직이는 상태에서 F키를 눌러 1인칭으로 변환한 상태이다.

- Dependencies

  Numpy, scipy를 사용(environment.yml에 작성)

- **실행법**

  기존과 동일. conda env create -f environment.yml 으로 가상환경 만든 후, conda activate snu_graphics로 가상환경 실행. 이후 python main.py로 pyglet window를 실행할 수 있음.

- 실행 화면

  Report에 동영상을 첨부하였음.