Lab 6: Animation, Time Series, and Dynamic Scenes

Learning Outcomes

By completing this lab, students will be able to:

- Animate 3D objects using time series data or frame-based logic.
- Create and visualize 2D and 3D vector fields that evolve over time.
- Apply and synchronize model transformations (translation, rotation, scaling) in animated scenes.
- Update 3D scenes dynamically in response to simulation or user-defined data.

Animation, Time Series, and Dynamic Scenes

Animation in 3D graphics brings time into the spatial domain, allowing data and geometry to evolve interactively or sequentially. Whether visualizing simulations, motion, or time-series datasets, animation provides an intuitive way to understand dynamic processes.

K3D allows animation through real-time updates of object attributes — such as positions, model matrices, colors, and vectors. Using Python loops or time-based data, users can simulate motion, apply transformations, or encode changing scalar values through color or shape.

Time series animation often uses temporal data to drive visual changes (e.g., a rising temperature mapped to vertex color). Meanwhile, vector field animation can simulate wind, flow, or force direction. Together, these techniques allow the creation of immersive, responsive, and insightful 3D dynamic scenes.

Animation in 3D graphics involves updating the properties of objects over time, such as position, rotation, scale, color, or shape. In this lab, we explore two primary forms of animation:

- Frame-based animation — updates geometry frame by frame using loops (e.g., rotating an object).
- Data-driven animation — maps changes in time-series or simulation data directly to 3D changes (e.g., a point cloud evolving with time).

Vector fields are particularly useful in physics and scientific visualization. A 2D or 3D vector field assigns a direction and magnitude (a vector) to every point in space. When animated, they simulate flows, forces, or motion patterns.

K3D allows real-time updates of object attributes such as positions, model_matrix, and color, enabling interactive dynamic scenes.

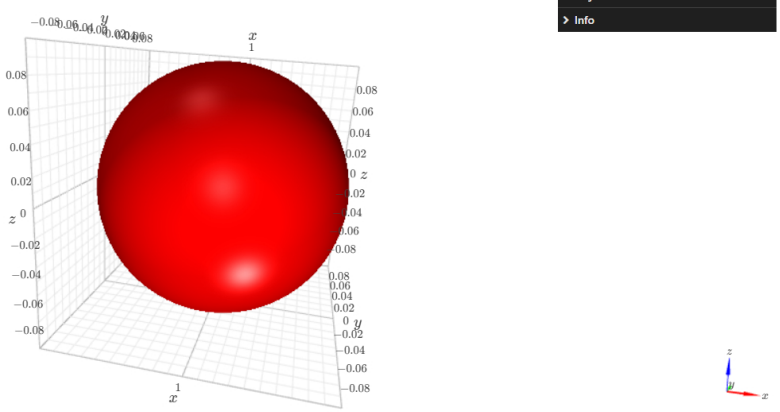
[ ]:
```
!pip install k3d
```

[17]:
```
import k3d
import numpy as np
import time
import math
```

Example 1: Animate a Moving Point Along a Circular Path

The following example demonstrates basic object animation using position updates over time.

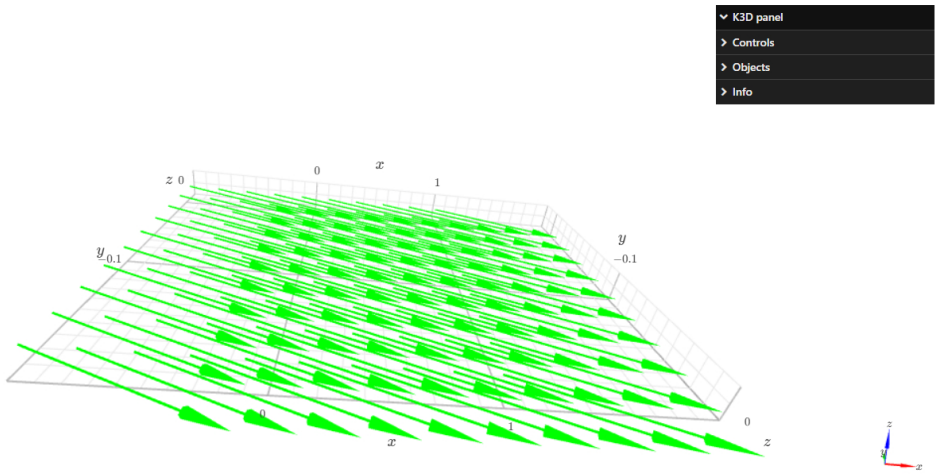
[7]:
```
plot = k3d.plot()

# Precompute a circular trajectory
N = 100
theta = np.linspace(0, 2 * np.pi, N)
trajectory = np.stack((np.cos(theta), np.sin(theta), np.zeros_like(theta)), axis=1)

# Create the moving point
point = k3d.points([[1, 0, 0]], point_size=0.2, color=0xff0000)
plot += point
plot.display()

# Animate the point along the path
for pos in trajectory:
    point.positions = np.array([pos], dtype=np.float32)
    time.sleep(0.03)
```



Example 2: Animate a 2D Vector Field

This example shows how vector fields can evolve over time using trigonometric direction updates.

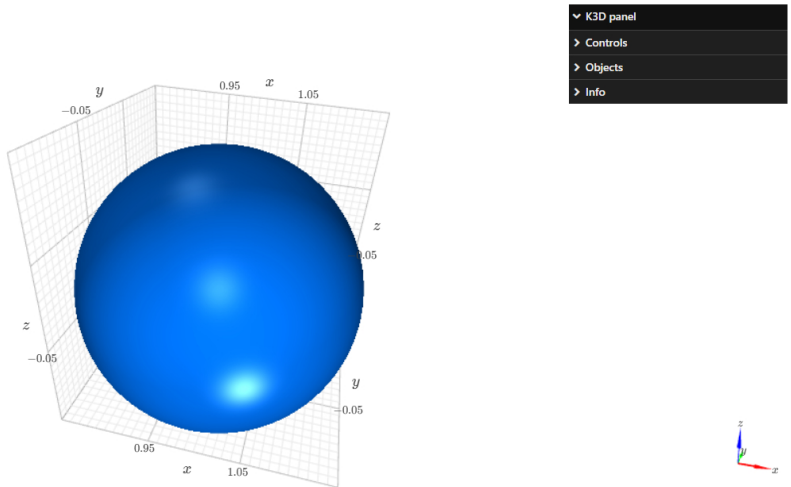
[10]:
```
plot = k3d.plot()

# Generate grid of positions
x, y = np.meshgrid(np.linspace(-1, 1, 10), np.linspace(-1, 1, 10))
origins = np.stack((x.ravel(), y.ravel(), np.zeros_like(x.ravel())), axis=1)

# Create initial directions
directions = np.stack((np.ones_like(x).ravel(), np.zeros_like(y).ravel(), np.zeros_like(x).ravel()), axis=1)

vectors = k3d.vectors(origins.astype(np.float32), directions.astype(np.float32), color=0x00ff00, line_width=0.01)
plot += vectors
plot.display()

# Animate: rotate all arrows
for t in range(60):
    angle = t * 0.1
    dx = np.cos(angle)
    dy = np.sin(angle)
    directions = np.stack((dx * np.ones_like(x).ravel(), dy * np.ones_like(y).ravel(), np.zeros_like(x).ravel()), axis=1)
    vectors.vectors = directions.astype(np.float32)
    time.sleep(0.05)
```



Example 3: Animate Transformation Using Model Matrix (Rotation)

This example applies animated transformation using 4×4 matrices (affine transformations).

[19]:
```
# Initial point
point = k3d.points([[1, 0, 0]], point_size=0.3, color=0x0077ff)
plot = k3d.plot()
plot += point
plot.display()

# Build rotating model matrices
for t in range(100):
    angle = 2 * math.pi * (t / 100)
    matrix = np.array([
        [math.cos(angle), -math.sin(angle), 0.0, 0.0],
        [math.sin(angle),  math.cos(angle), 0.0, 0.0],
        [0.0,              0.0,             1.0, 0.0],
        [0.0,              0.0,             0.0, 1.0]
    ], dtype=np.float32)

    point.model_matrix = matrix
    time.sleep(0.05)
```

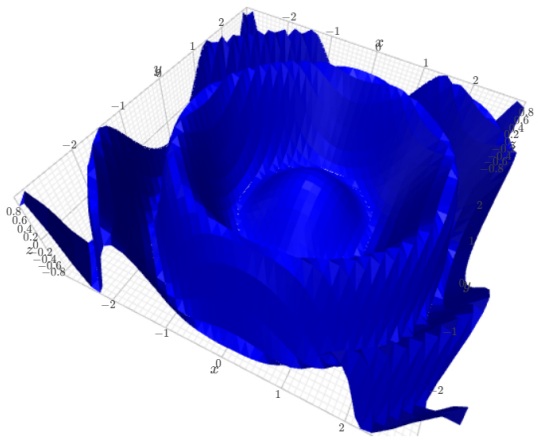


Example 4: Animate a Scalar Field as Surface Waves

This example shows how animated surface data can simulate physical phenomena like waves.

[22]:
```
plot = k3d.plot()

# Define spatial grid
x, y = np.meshgrid(np.linspace(-3, 3, 40), np.linspace(-3, 3, 40))
res = x.shape[0]
faces = []
for i in range(res - 1):
    for j in range(res - 1):
        idx = i * res + j
        faces.append([idx, idx + 1, idx + res])
        faces.append([idx + 1, idx + res + 1, idx + res])
faces = np.array(faces, dtype=np.uint32)

# Initial Z surface
z = np.sin(x**2 + y**2)
vertices = np.stack([x.ravel(), y.ravel(), z.ravel()], axis=1)
mesh = k3d.mesh(vertices.astype(np.float32), indices=faces)
plot += mesh
plot.display()

# Animate: wave propagation
for t in range(80):
    z = np.sin(x**2 + y**2 - t * 0.2)
    vertices = np.stack([x.ravel(), y.ravel(), z.ravel()], axis=1)
    mesh.vertices = vertices.astype(np.float32)
    time.sleep(0.05)
```

K3D panel
Controls
Objects
Info



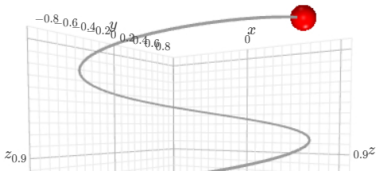
Example 5: Animate Object Along Predefined Path (With Trail)

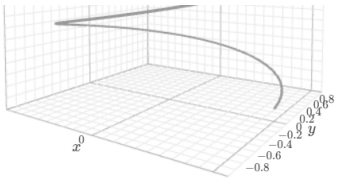
This example combines path-following with visual trail for storytelling or data tracing.

[29]:
```
plot = k3d.plot()

# Predefined path (spiral)
t_vals = np.linspace(0, 4 * np.pi, 100)
x = np.cos(t_vals)
y = np.sin(t_vals)
z = t_vals / (2 * np.pi)

path = np.stack((x, y, z), axis=1)
trail = k3d.line(path, width=0.02, color=0x999999)

point = k3d.points([[x[0], y[0], z[0]]], point_size=0.2, color=0xff0000)

plot += trail
plot += point
plot.display()

# Animate the point along the spiral
for i in range(len(path)):
    point.positions = np.array([path[i]], dtype=np.float32)
    time.sleep(0.04)
```

```
C:\Users\arija\anaconda3\Lib\site-packages\traittypes\traittypes.py:97: UserWarning: Given trait value dtype "float64" does not match required type "float32". A coerced copy has been created.
  warnings.warn(
```

K3D panel
Controls
Objects
Info

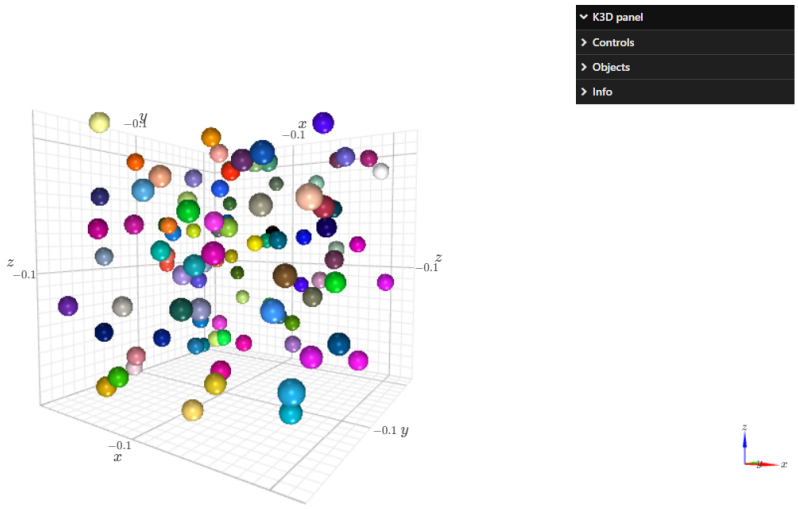




Example 6: Animate Time Series with Changing Point Colors

This example visualizes a time-dependent signal or measurement (like temperature, activity, or pressure) via color.

[31]:
```
plot = k3d.plot()

# Create points with fixed positions
positions = np.random.uniform(-1, 1, (100, 3)).astype(np.float32)
points = k3d.points(positions, point_size=0.15)
plot += points
plot.display()

# Animate color change based on simulated data values
for t in range(80):
    values = np.sin(np.linspace(0, 2*np.pi, 100) + t * 0.2)
    norm = ((values - values.min()) / values.ptp()) * 0xFFFFFF
    points.colors = norm.astype(np.uint32)
    time.sleep(0.05)
```



Tasks

Task 1: Animate a 3D Object Along a Custom Path

Move a 3D object (e.g., a sphere or cube) along a non-circular path such as a figure-eight or zig-zag, and visualize its movement. Steps:

- Define the path as a series of (x, y, z) points.
- Use k3d.points() to represent the object.
- Update its position in a loop using .positions.

An outcome of this should be a moving object that traces a recognizable path in 3D space.

---

[44]:
```
## Write your ode here
```

Task 2: Simulate a Wind Field Using Animated 3D Vectors

Create a 3D grid of vectors and animate their directions to simulate wind or flow. Steps:

- Define a grid of vector origins.
- Animate vector directions over time using sine or cosine functions.
- Use k3d.vectors() and update .vectors dynamically.

An outcome of this should be an evolving vector field that shows dynamic direction changes.

---

[46]:
```
## Write your ode here
```

Task 3: Animate Temperature Across a Surface Mesh

Create a surface (e.g., terrain or sine wave), and simulate changing "temperature" by animating vertex colors. Steps:

- Generate a 2D surface mesh.
- Create a temperature wave that evolves over time (sinusoidal or noisy).
- Convert temperature values to colors and update .colors.

An outcome of this task is a surface mesh with a flowing heat map effect.

---

[48]:
```
## Write your ode here
```

Task 4: Build a 3D Time Series Dashboard

Display changing data values using position, size, or color of bars or points — like an animated chart. Steps:

- Create vertical lines or points to represent data values.
- Use a loop to simulate new data over time.
- Animate the scene using .positions, .colors, or .model_matrix.

Outcome is a 3D dashboard that visually tracks change — ideal for performance metrics or scientific data playback.

---

[50]:
```
## Write your ode here
```