Lab 5: Textures and Materials in 3D

This lab focuses on the integration of texture and textual information into 3D scenes using K3D. Students will learn how to apply 2D images (bitmaps) as textures on 3D surfaces, enabling more realistic or informative visualizations. They will also explore how to embed text into scenes for annotation or labeling — both as flat 2D overlays and as extruded 3D objects. In addition, procedural approaches will be introduced to simulate surface variation and material properties, even without image-based textures.

Through hands-on examples, students will develop an understanding of how visual appearance can be enhanced using textures and text, contributing to both the readability and realism of 3D visualizations.

Learning Objectives

By the end of this lab, students will be able to:

- Apply 2D bitmap textures to 3D objects in K3D.
- Use text labels as visual annotations or scene elements.
- Experiment with text rendering as both flat overlays and 3D geometries.
- Understand how material appearance is affected by color, lighting, and texture mapping.

Textures and Materials in 3D Graphics

Textures are 2D images (bitmaps) that are mapped onto 3D geometry to simulate surface detail, patterns, or annotations. They replace or supplement simple color assignments by adding visual richness without increasing geometric complexity. Textures can be photographic, procedurally generated, or text-based.

Material appearance refers to how objects interact with light — through color, reflectivity, texture, and transparency. In real-time 3D rendering, materials are often simplified but can include texture maps (for color, bump, or normal effects) to simulate realism.

Text rendering in 3D graphics can be done in two ways:

- 2D overlay text (text2d) which always faces the camera and behaves like a screen label.
- 3D text geometry (text) which exists as part of the 3D scene and responds to perspective and transformation.

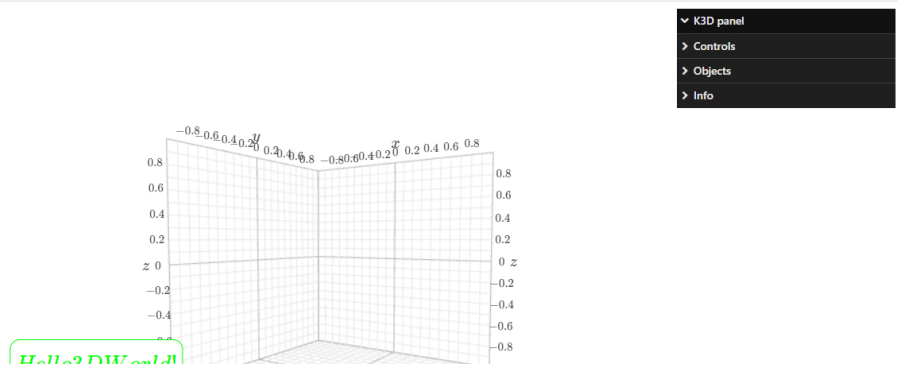
[7]:
```
!pip install k3d
```

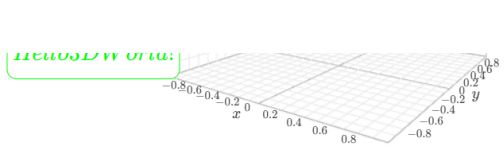
```
Requirement already satisfied: k3d in c:\users\arija\anaconda3\lib\site-packages (2.16.1)
Requirement already satisfied: ipywidgets<9.0.0,>=7.0.0 in c:\users\arija\anaconda3\lib\site-packages (from k3d) (8.1.5)
Requirement already satisfied: msgpack in c:\users\arija\anaconda3\lib\site-packages (from k3d) (1.0.3)
Requirement already satisfied: numpy in c:\users\arija\anaconda3\lib\site-packages (from k3d) (1.26.4)
Requirement already satisfied: traitlets in c:\users\arija\anaconda3\lib\site-packages (from k3d) (5.14.3)
Requirement already satisfied: traittypes in c:\users\arija\anaconda3\lib\site-packages (from k3d) (0.2.1)
Requirement already satisfied: comm>=0.1.3 in c:\users\arija\anaconda3\lib\site-packages (from ipywidgets<9.0.0,>=7.0.0->k3d) (0.2.1)
Requirement already satisfied: ipython>=6.1.0 in c:\users\arija\anaconda3\lib\site-packages (from ipywidgets<9.0.0,>=7.0.0->k3d) (8.27.0)
Requirement already satisfied: widgetsnbextension~=4.0.12 in c:\users\arija\anaconda3\lib\site-packages (from ipywidgets<9.0.0,>=7.0.0->k3d) (4.0.13)
Requirement already satisfied: jupyterlab-widgets~=3.0.12 in c:\users\arija\anaconda3\lib\site-packages (from ipywidgets<9.0.0,>=7.0.0->k3d) (3.0.13)
Requirement already satisfied: decorator in c:\users\arija\anaconda3\lib\site-packages (from ipython>=6.1.0->ipywidgets<9.0.0,>=7.0.0->k3d) (5.1.1)
Requirement already satisfied: jedi>=0.16 in c:\users\arija\anaconda3\lib\site-packages (from ipython>=6.1.0->ipywidgets<9.0.0,>=7.0.0->k3d) (0.19.1)
Requirement already satisfied: matplotlib-inline in c:\users\arija\anaconda3\lib\site-packages (from ipython>=6.1.0->ipywidgets<9.0.0,>=7.0.0->k3d) (0.1.6)
Requirement already satisfied: prompt-toolkit<3.1.0,>=3.0.41 in c:\users\arija\anaconda3\lib\site-packages (from ipython>=6.1.0->ipywidgets<9.0.0,>=7.0.0->
k3d) (3.0.43)
Requirement already satisfied: pygments>=2.4.0 in c:\users\arija\anaconda3\lib\site-packages (from ipython>=6.1.0->ipywidgets<9.0.0,>=7.0.0->k3d) (2.15.1)
Requirement already satisfied: stack-data in c:\users\arija\anaconda3\lib\site-packages (from ipython>=6.1.0->ipywidgets<9.0.0,>=7.0.0->k3d) (0.2.0)
Requirement already satisfied: colorama in c:\users\arija\anaconda3\lib\site-packages (from ipython>=6.1.0->ipywidgets<9.0.0,>=7.0.0->k3d) (0.4.6)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in c:\users\arija\anaconda3\lib\site-packages (from jedi>=0.16->ipython>=6.1.0->ipywidgets<9.0.0,>=7.0.0
->k3d) (0.8.3)
Requirement already satisfied: wcwidth in c:\users\arija\anaconda3\lib\site-packages (from prompt-toolkit<3.1.0,>=3.0.41->ipython>=6.1.0->ipywidgets<9.0.0,
>=7.0.0->k3d) (0.2.5)
Requirement already satisfied: executing in c:\users\arija\anaconda3\lib\site-packages (from stack-data->ipython>=6.1.0->ipywidgets<9.0.0,>=7.0.0->k3d) (0.
8.3)
Requirement already satisfied: asttokens in c:\users\arija\anaconda3\lib\site-packages (from stack-data->ipython>=6.1.0->ipywidgets<9.0.0,>=7.0.0->k3d) (2.
0.5)
Requirement already satisfied: pure-eval in c:\users\arija\anaconda3\lib\site-packages (from stack-data->ipython>=6.1.0->ipywidgets<9.0.0,>=7.0.0->k3d) (0.
2.2)
Requirement already satisfied: six in c:\users\arija\anaconda3\lib\site-packages (from asttokens->stack-data->ipython>=6.1.0->ipywidgets<9.0.0,>=7.0.0->k3
d) (1.16.0)
```

Example 1: Add a 2D Text Label (Overlay)

This places a fixed 2D label in the corner of the screen — great for titles or dynamic overlays.

[16]:
```
import k3d

plot = k3d.plot()
label = k3d.text2d("Hello 3D World!", position=[0.2, 0.65], size=1.5, color=0x00ff00)
plot += label
plot.display()
```

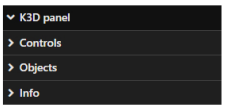


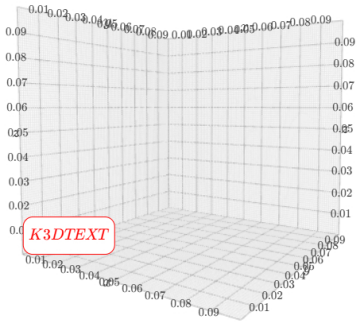


Example 2: Add a 3D Text Object into the Scene

This creates a 3D object that behaves like any mesh — you can rotate, scale, and position it in 3D space.

[20]:
```
text = k3d.text("K3D TEXT", position=[0, 0, 0], scale=1.0, color=0xff0000)
plot = k3d.plot()
plot += text
plot.display()
```

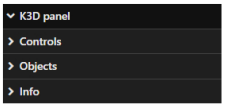


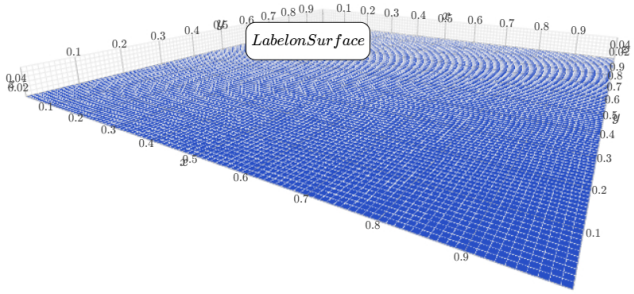


Example 3: Display a Colored Surface with Text Label

Combines a surface mesh and a 3D text object to annotate geometry in the scene.

[25]:
```
import numpy as np

vertices = np.array([
    [0, 0, 0],
    [1, 0, 0],
    [1, 1, 0],
    [0, 1, 0]
], dtype=np.float32)

indices = np.array([[0, 1, 2], [0, 2, 3]], dtype=np.uint32)

plot = k3d.plot()
mesh = k3d.mesh(vertices, indices, color=0x3366ff)
plot += mesh
plot += k3d.text("Label on Surface", position=[0.3, 0.3, 0.05], scale=0.3, color=0x000000)
plot.display()
```



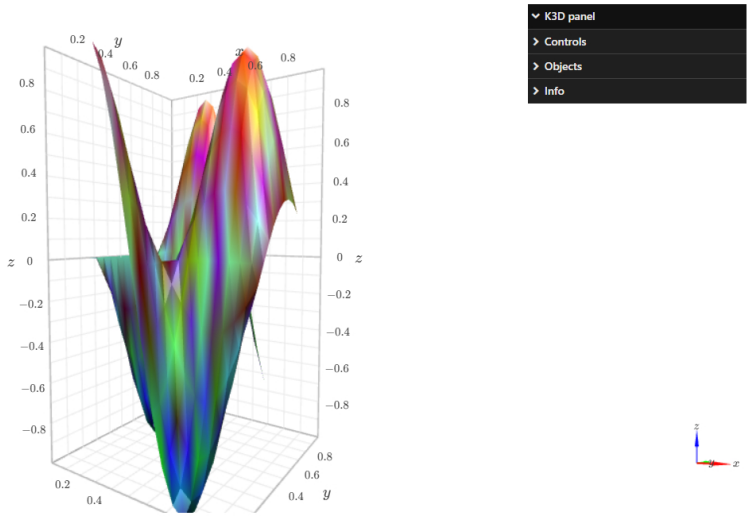


Example 4: Apply Per-Vertex Color Gradient (Procedural Texture)

Demonstrates procedural "texture" using vertex color mapping instead of image-based texturing.

[28]:
```
x, y = np.meshgrid(np.linspace(0, 1, 20), np.linspace(0, 1, 20))
z = np.sin(x * 6) * np.cos(y * 6)
vertices = np.stack((x.ravel(), y.ravel(), z.ravel()), axis=1)
```

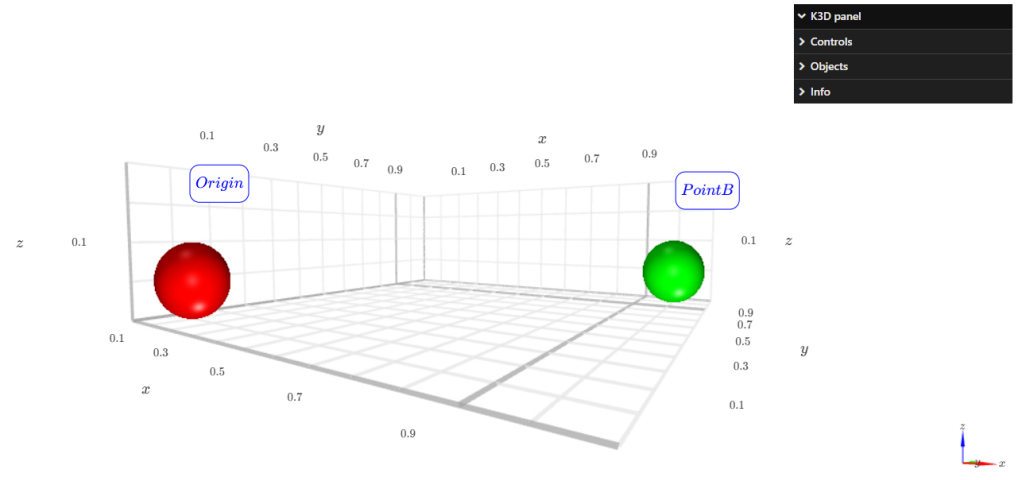
```
indices = []
res = x.shape[0]
for i in range(res - 1):
    for j in range(res - 1):
        idx = i * res + j
        indices.append([idx, idx + 1, idx + res])
        indices.append([idx + 1, idx + res + 1, idx + res])
indices = np.array(indices, dtype=np.uint32)

# Color by height (Z)
z_vals = z.ravel()
norm_z = (z_vals - z_vals.min()) / z_vals.ptp()
colors = (norm_z * 0xFFFFFF).astype(np.uint32)

plot = k3d.plot()
plot += k3d.mesh(vertices=vertices.astype(np.float32), indices=indices, colors=colors)
plot.display()
```



Example 5: Add Multiple 3D Text Labels to Objects

Shows how 3D text can be dynamically positioned near points or regions of interest.

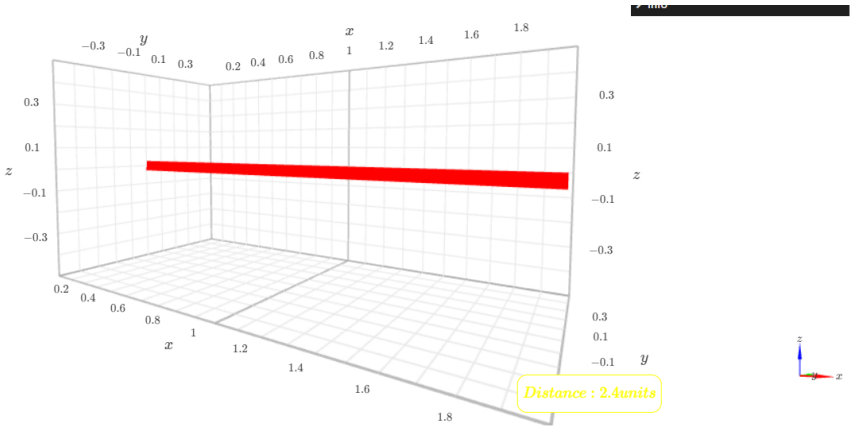
[31]:
```
plot = k3d.plot()

# Two spheres
plot += k3d.points([[0, 0, 0]], point_size=0.2, color=0xff0000)
plot += k3d.points([[1, 1, 0]], point_size=0.2, color=0x00ff00)

# Add text labels
plot += k3d.text("Origin", position=[0, 0, 0.2], scale=0.2)
plot += k3d.text("Point B", position=[1, 1, 0.2], scale=0.2)

plot.display()
```



Example 6: Create a 2D Overlay for Real-Time Measurement Display

This simulates a simple measurement or HUD-style interface in 3D scenes.

[34]:
```
plot = k3d.plot()
plot += k3d.text2d("Distance: 2.4 units", position=[0.7, 0.9], size=1.0, color=0xffff00)
plot += k3d.line([[0, 0, 0], [2, 0, 0]], width=0.05, color=0xff0000)
plot.display()
```

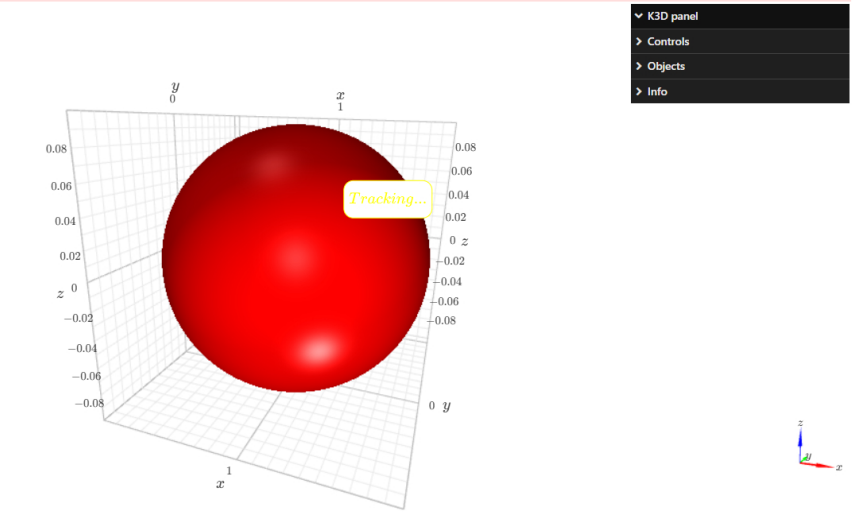


Example 7: Animated Labels for Moving Objects

Create a dynamic system where 3D objects move, and a text label updates its position to follow one of them in real time.

[38]:
```
import k3d
import numpy as np
import time

plot = k3d.plot()

# Create a point that will move in a circular orbit
N = 100
theta = np.linspace(0, 2 * np.pi, N)
orbit = np.stack((np.cos(theta), np.sin(theta), np.zeros_like(theta)), axis=1)

# Initialize point and label
point = k3d.points(positions=np.array([[1, 0, 0]]), point_size=0.2, color=0xff0000)
label = k3d.text("Tracking...", position=[1, 0.1, 0], scale=0.15, color=0xffff00)

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

# Animate object + label
for i in range(N):
    pos = orbit[i]
    point.positions = np.array([pos], dtype=np.float32)
    label.position = (pos + np.array([0, 0.15, 0])).tolist()
    time.sleep(0.05)
```

```
C:\Users\arija\anaconda3\Lib\site-packages\traittypes\traittypes.py:97: UserWarning: Given trait value dtype "int32" does not match required type "float32". A coerced copy has been created.
  warnings.warn(
```



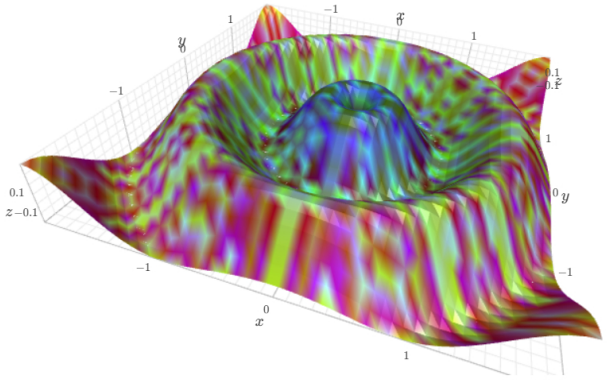
Example 8: Simulated Material Effects with Procedural Coloring

Mimic material surface variation (e.g. roughness or terrain elevation) using procedural vertex coloring based on curvature-like features.

[44]:
```
# Create a surface using radial sine wave
x, y = np.meshgrid(np.linspace(-2, 2, 50), np.linspace(-2, 2, 50))
r = np.sqrt(x**2 + y**2)
z = 0.3 * np.sin(5 * r)

vertices = np.stack((x.ravel(), y.ravel(), z.ravel()), axis=1)

# Generate triangles
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)
```

```
# Simulate material: color based on distance from origin
dists = np.sqrt(vertices[:, 0]**2 + vertices[:, 1]**2)
dists_norm = (dists - dists.min()) / dists.ptp()
colors = (dists_norm * 0xAAAAAA + 0x333333).astype(np.uint32)  # dark inner, light outer

plot = k3d.plot()
plot += k3d.mesh(vertices=vertices.astype(np.float32), indices=faces, colors=colors)
plot.display()
```

K3D panel
Controls
Objects
Info

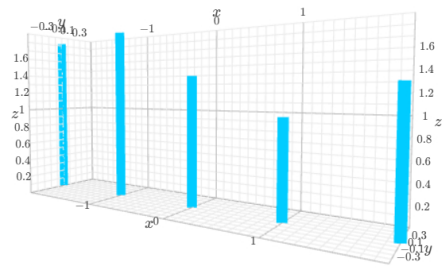


Example 9: Create a 3D Info Visualization Dashboard

Create a simple data dashboard using bars (as geometry) and text labels to represent values in 3D space.

[47]:
```
plot = k3d.plot()

# Simulated "bar chart" in 3D
positions = np.linspace(-2, 2, 5)
heights = np.random.uniform(0.5, 2.0, size=5)

for i, (x, h) in enumerate(zip(positions, heights)):
    # Create a vertical line as a bar
    bar = k3d.line([[x, 0, 0], [x, 0, h]], width=0.1, color=0x00ccff)
    label = k3d.text2d(f"Val {i+1}: {h:.2f}", position=[0.05, 0.9 - i * 0.07], size=1.2)

    plot += bar
    plot += label

plot.display()
```

K3D panel
Controls
Objects
Info



Val5 : 1.30
Val4 : 0.98
Val3 : 1.38
Val2 : 1.90
Val1 : 1.82

Tasks

Task 1: Create a Text-Labeled Terrain

Combine a procedurally generated terrain (e.g. sine-based) with several 3D text labels placed at key regions (e.g. peaks or valleys). Steps:

- Create a mesh with z = sin(x) * cos(y).
- Place text objects near the highest and lowest points.
- Use color to indicate elevation.

Task 2: Simulate a Control Panel Using 2D Overlays

Use text2d elements to display interface elements such as coordinates, frame rate, or simulation info. Steps:

- Create a 3D object (e.g. rotating cube or orbiting point).
- Add 2D overlay text that updates or labels information.

Task 3: Procedural Color Texture on a Curved Surface

Visualize a mathematical surface (e.g. paraboloid or sine wave) with a color gradient based on height. Steps:

- Generate mesh and assign per-vertex color.
- Use normalization + color encoding to simulate texture.
- Compare with flat color.

Task 4: Label a 3D Point Cloud or Model

Load or create a point cloud and annotate selected points with 3D text. Steps:

- Generate or import a 3D point set.
- Add 3D text near selected or randomly chosen points.
- Group related points with the same label color.

Task 5: Create a 3D Labeled Coordinate System Viewer

Build a labeled 3D coordinate reference frame with axis lines and corresponding text labels for X, Y, and Z. Add arrows or points to enhance orientation. Steps:

- Use k3d.line() to create three colored axes: X (red), Y (green), Z (blue)
- Add k3d.text() labels near the end of each axis: "X", "Y", and "Z".
- Place small k3d.points() or arrows on the axis ends.
- Style everything for clarity: consistent colors, readable scales, centered origin.

The outcome should be a reusable 3D coordinate reference tool that can be included in any future visualizations or used as an orientation helper.

Task 6: Build a Scene Legend Using Floating 2D Labels

Create a floating legend using multiple text2d elements in the corner of the screen, describing elements in your 3D scene (e.g., red = error, green = safe, blue = neutral). Steps:

- Create 3D objects with different colors (e.g., spheres, points, bars).
- Place text2d() labels in the top-right of the screen to describe what each color means.
- Add a mini title (Scene Legend) in bold or larger size.
- Use consistent formatting and positions (e.g., [0.7, 0.9], [0.7, 0.85], etc.)

The outcome shpuld be an interactive 3D visualization with an integrated floating legend to help users interpret the scene intuitively — useful for dashboards, reports, or presentation visuals.
