KITTI 3D Object Detection - Data Exploration and Visualization

This notebook provides comprehensive data exploration, visualization, and analysis for the KITTI 3D object detection dataset.

In [1]:
```
import os
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.cluster import KMeans, DBSCAN
from sklearn.preprocessing import StandardScaler
import torch
from config import cfg
from dataset import KITTIDataset, build_transforms

# Set style
sns.set_style('whitegrid')
plt.rcParams['figure.figsize'] = (12, 8)

print("Libraries imported successfully!")
```

```
Libraries imported successfully!
```

1. Dataset Overview

In [2]:
```
# Load dataset
dataset = KITTIDataset(
    root=cfg.DATA_ROOT,
    split="training",
    transform=None
)

print(f"Total samples: {len(dataset)}")
print(f"Image directory: {dataset.image_dir}")
print(f"Label directory: {dataset.label_dir}")
```

```
Total samples: 7481
Image directory: C:\Users\94675\Desktop\group_project_pro\data/kitti\training\ima
ge_2
Label directory: C:\Users\94675\Desktop\group_project_pro\data/kitti\training\tra
ining\label_2
```

2. Data Distribution Analysis

In [3]:
```
# Collect all 3D box parameters
all_boxes = []
for i in range(len(dataset)):
    _, target, _ = dataset[i]
```

```
    if target.sum() != 0:  # Skip empty targets
        all_boxes.append(target.numpy())

all_boxes = np.array(all_boxes)
print(f"Valid samples with Car objects: {len(all_boxes)}")

# Create DataFrame
df = pd.DataFrame(all_boxes, columns=['x', 'y', 'z', 'length', 'width', 'height'
df.head()
```

```
Valid samples with Car objects: 6684
```

Out[3]:

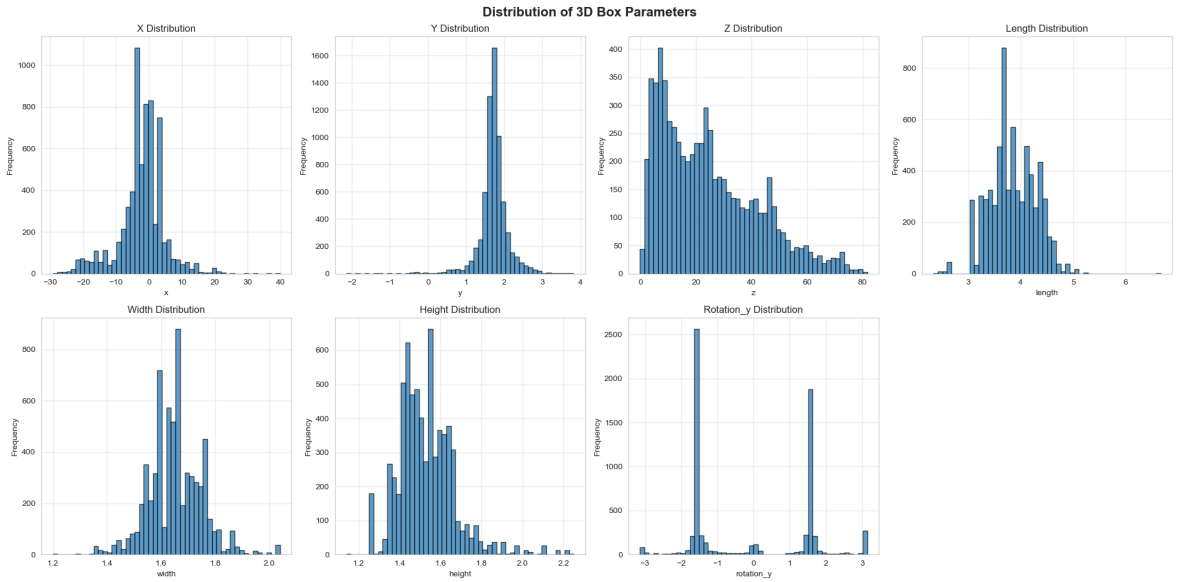
	x	y	z	length	width	height	rotation_y
0	-16.530001	2.39	58.490002	3.69	1.87	1.67	1.57
1	3.180000	2.27	34.380001	4.36	1.58	1.41	-1.58
2	1.000000	1.75	13.220000	4.15	1.73	1.57	1.62
3	-15.710000	2.16	38.259998	4.01	1.76	1.49	1.57
4	-2.720000	0.82	48.220001	3.62	1.56	1.48	-1.62

In [4]:
```
# Statistical summary
print("\n=== Statistical Summary ===")
print(df.describe())
```

```
=== Statistical Summary ===
                 x            y            z       length        width  \
count  6684.000000  6684.000000  6684.000000  6684.000000  6684.000000
mean     -2.248752     1.725507    24.496105     3.839042     1.652370
std       6.829262     0.375542    17.427032     0.441941     0.099436
min     -29.180000    -2.140000    -0.180000     2.330000     1.200000
25%      -4.570000     1.600000     9.597500     3.560000     1.590000
50%      -1.695000     1.720000    21.315000     3.830000     1.650000
75%       1.150000     1.870000    35.830002     4.170000     1.720000
max      39.860001     3.820000    81.660004     6.670000     2.040000

            height   rotation_y
count  6684.000000  6684.000000
mean      1.530964    -0.111287
std       0.138064     1.668206
min       1.140000    -3.140000
25%       1.440000    -1.580000
50%       1.510000    -1.290000
75%       1.610000     1.570000
max       2.250000     3.140000
```

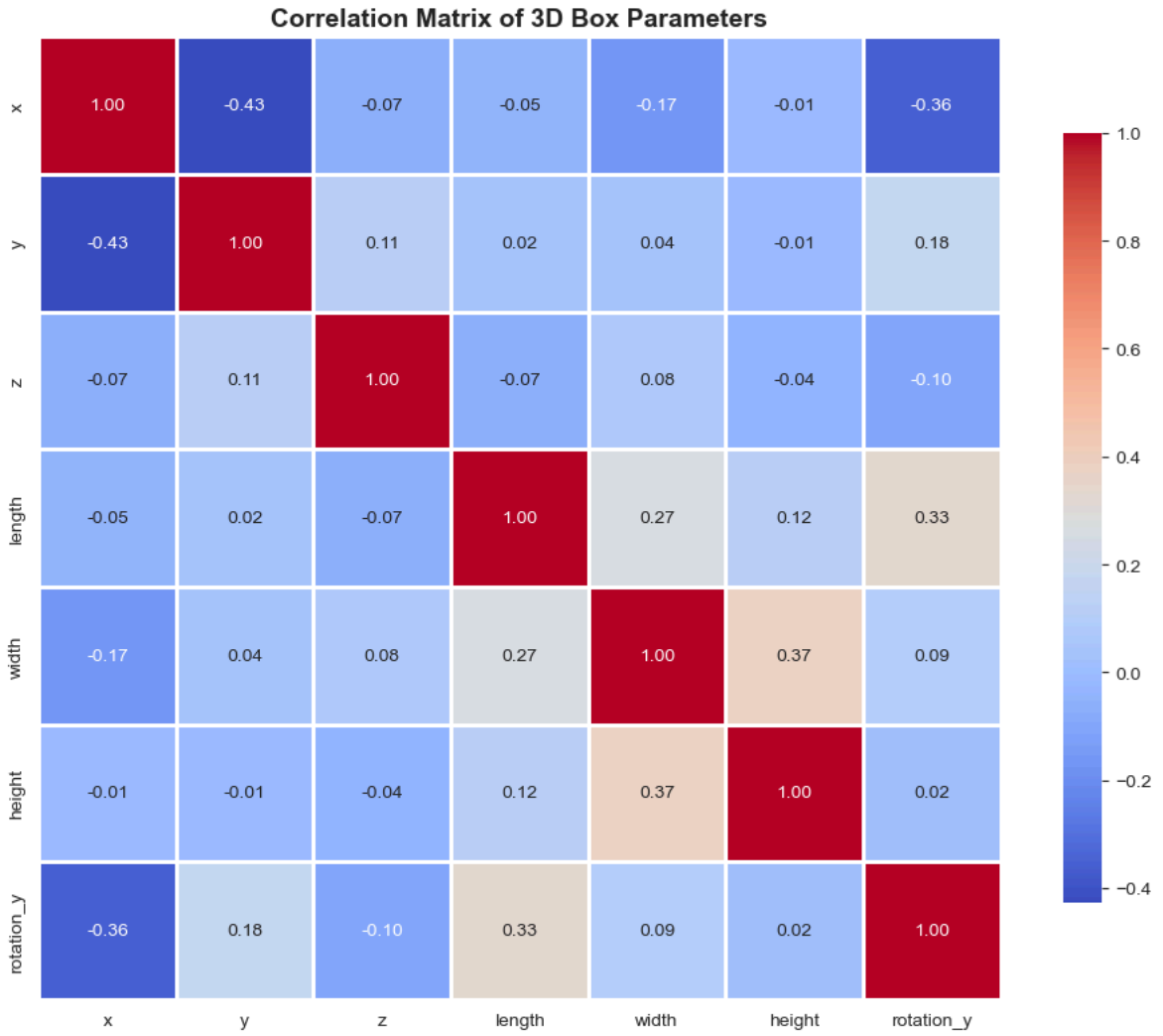
In [5]:
```
# Distribution plots
fig, axes = plt.subplots(2, 4, figsize=(20, 10))
fig.suptitle('Distribution of 3D Box Parameters', fontsize=16, fontweight='bold'

columns = ['x', 'y', 'z', 'length', 'width', 'height', 'rotation_y']
for idx, col in enumerate(columns):
    row = idx // 4
    col_idx = idx % 4
    axes[row, col_idx].hist(df[col], bins=50, edgecolor='black', alpha=0.7)
    axes[row, col_idx].set_title(f'{col.capitalize()} Distribution')
    axes[row, col_idx].set_xlabel(col)
    axes[row, col_idx].set_ylabel('Frequency')
    axes[row, col_idx].grid(True, alpha=0.3)
```

```
# Remove empty subplot
fig.delaxes(axes[1, 3])
plt.tight_layout()
plt.savefig(os.path.join(cfg.VIS_DIR, 'parameter_distributions.png'), dpi=300, b
plt.show()
```



3. Correlation Analysis

In [6]:
```
# Correlation matrix
plt.figure(figsize=(10, 8))
correlation_matrix = df.corr()
sns.heatmap(correlation_matrix, annot=True, fmt='.2f', cmap='coolwarm',
            square=True, linewidths=1, cbar_kws={"shrink": 0.8})
plt.title('Correlation Matrix of 3D Box Parameters', fontsize=14, fontweight='bo
plt.tight_layout()
plt.savefig(os.path.join(cfg.VIS_DIR, 'correlation_matrix.png'), dpi=300, bbox_i
plt.show()
```

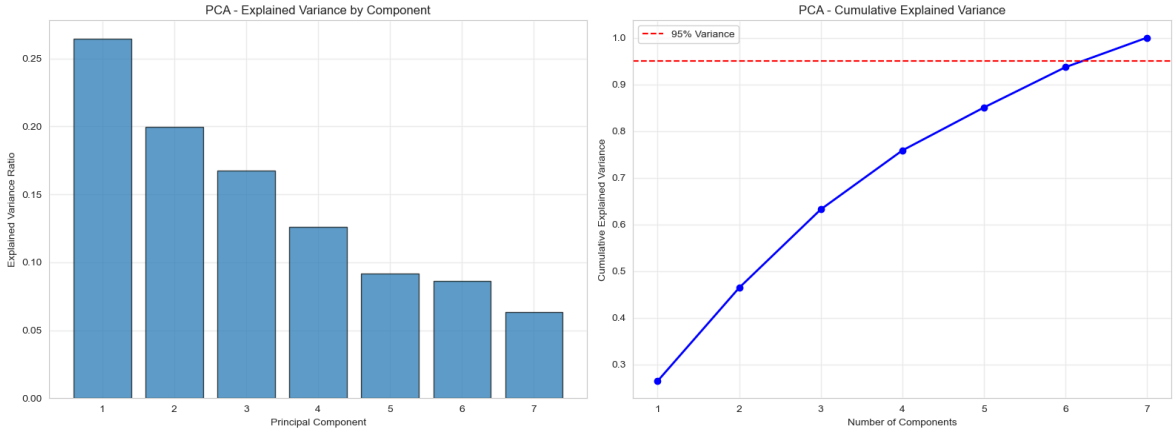


4. Dimensionality Reduction - PCA

In [7]:

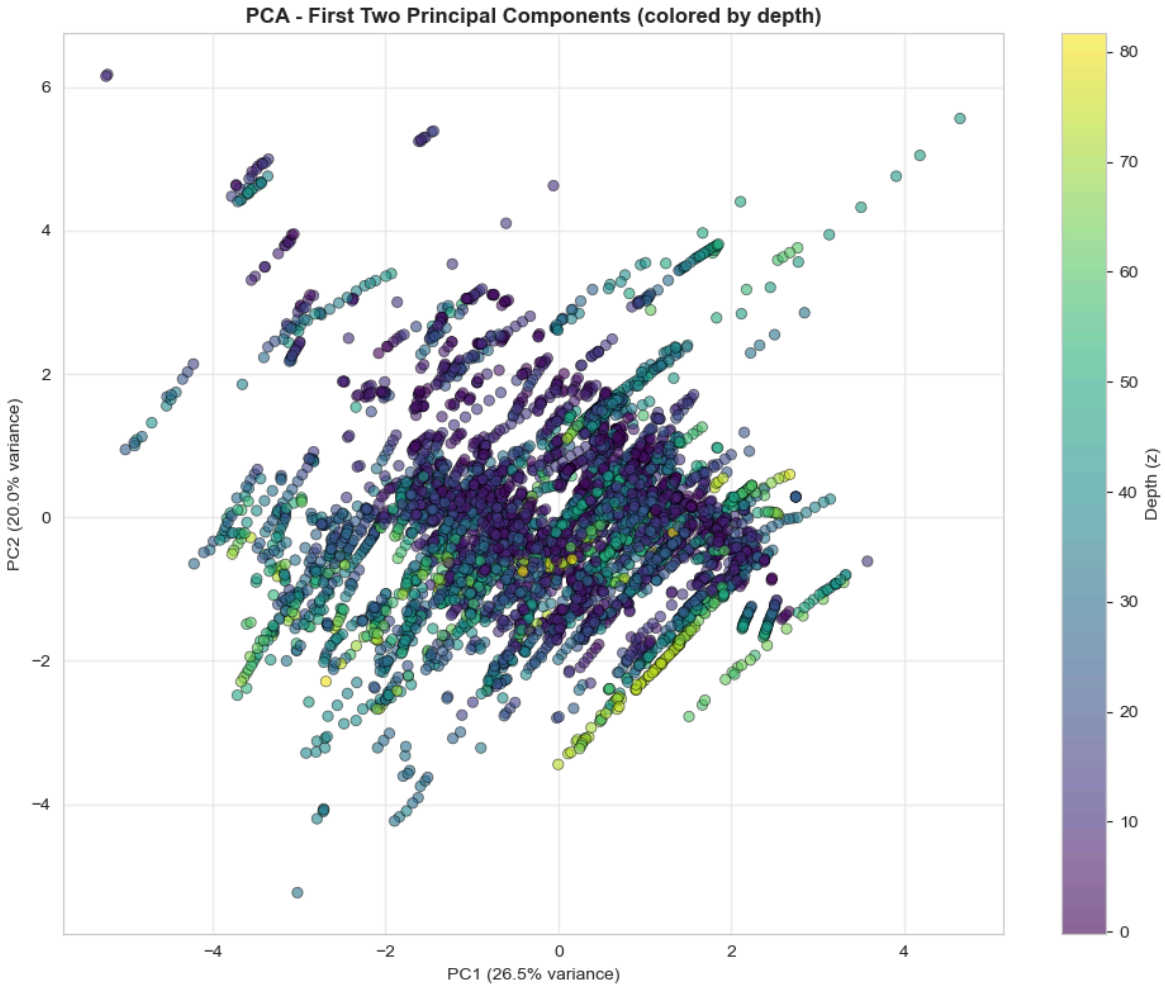
```
# Standardize data
scaler = StandardScaler()
data_scaled = scaler.fit_transform(df)

# Apply PCA
pca = PCA()
pca_result = pca.fit_transform(data_scaled)

# Explained variance
explained_variance = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance)

# Plot explained variance
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

ax1.bar(range(1, len(explained_variance)+1), explained_variance, alpha=0.7, edge
ax1.set_xlabel('Principal Component')
ax1.set_ylabel('Explained Variance Ratio')
ax1.set_title('PCA - Explained Variance by Component')
ax1.grid(True, alpha=0.3)

ax2.plot(range(1, len(cumulative_variance)+1), cumulative_variance, 'bo-', linew
ax2.axhline(y=0.95, color='r', linestyle='--', label='95% Variance')
ax2.set_xlabel('Number of Components')
```

```
ax2.set_ylabel('Cumulative Explained Variance')
ax2.set_title('PCA - Cumulative Explained Variance')
ax2.legend()
ax2.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig(os.path.join(cfg.VIS_DIR, 'pca_variance.png'), dpi=300, bbox_inches=
plt.show()

print(f"\nVariance explained by first 3 components: {cumulative_variance[2]:.2%}
```




```
Variance explained by first 3 components: 63.21%
```

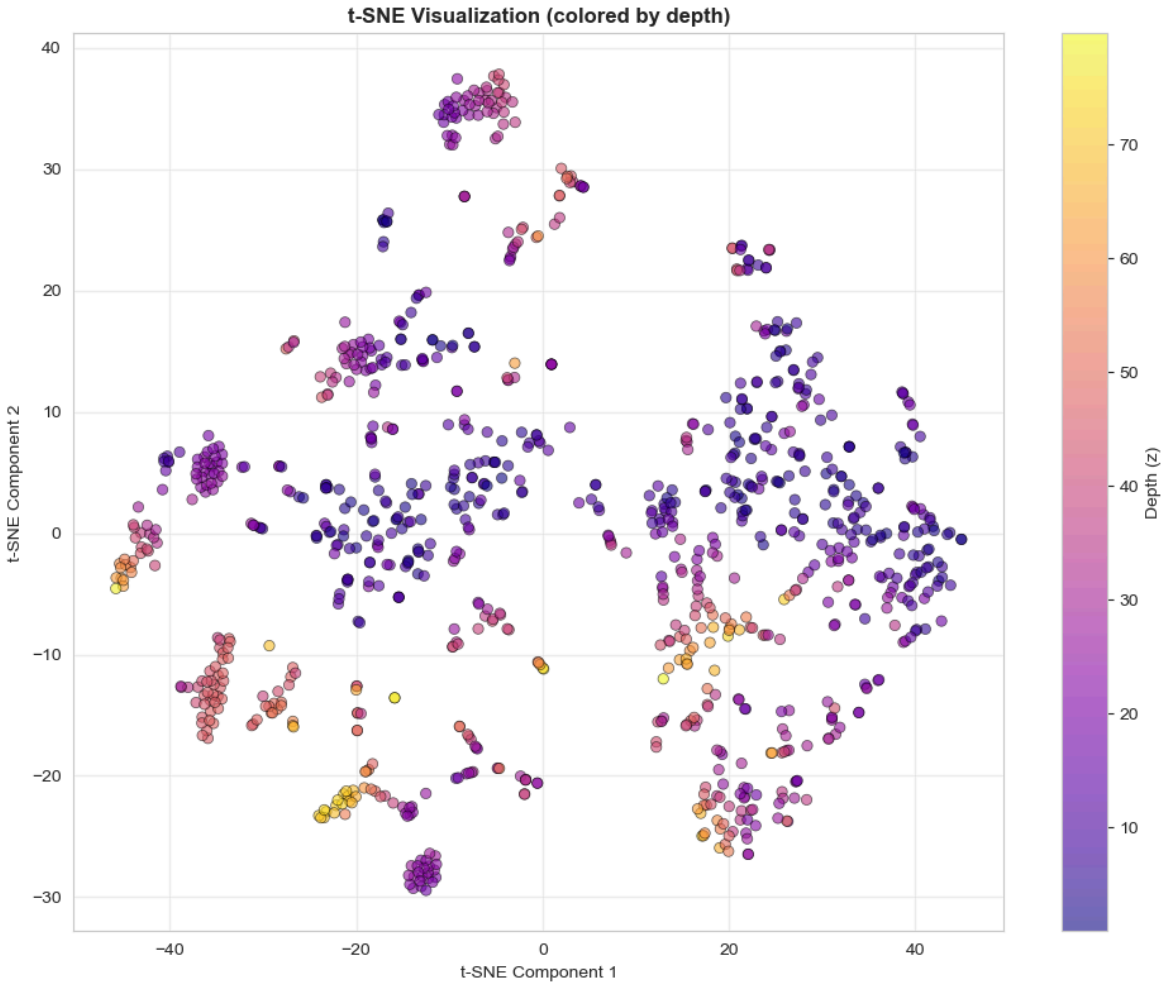
In [8]:
```
# 2D PCA visualization
plt.figure(figsize=(10, 8))
scatter = plt.scatter(pca_result[:, 0], pca_result[:, 1],
                      c=df['z'], cmap='viridis', alpha=0.6, edgecolors='black', l
plt.colorbar(scatter, label='Depth (z)')
plt.xlabel(f'PC1 ({explained_variance[0]:.1%} variance)')
plt.ylabel(f'PC2 ({explained_variance[1]:.1%} variance)')
plt.title('PCA - First Two Principal Components (colored by depth)', fontweight=
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig(os.path.join(cfg.VIS_DIR, 'pca_2d.png'), dpi=300, bbox_inches='tight
plt.show()
```



5. Dimensionality Reduction - t-SNE

In [10]:

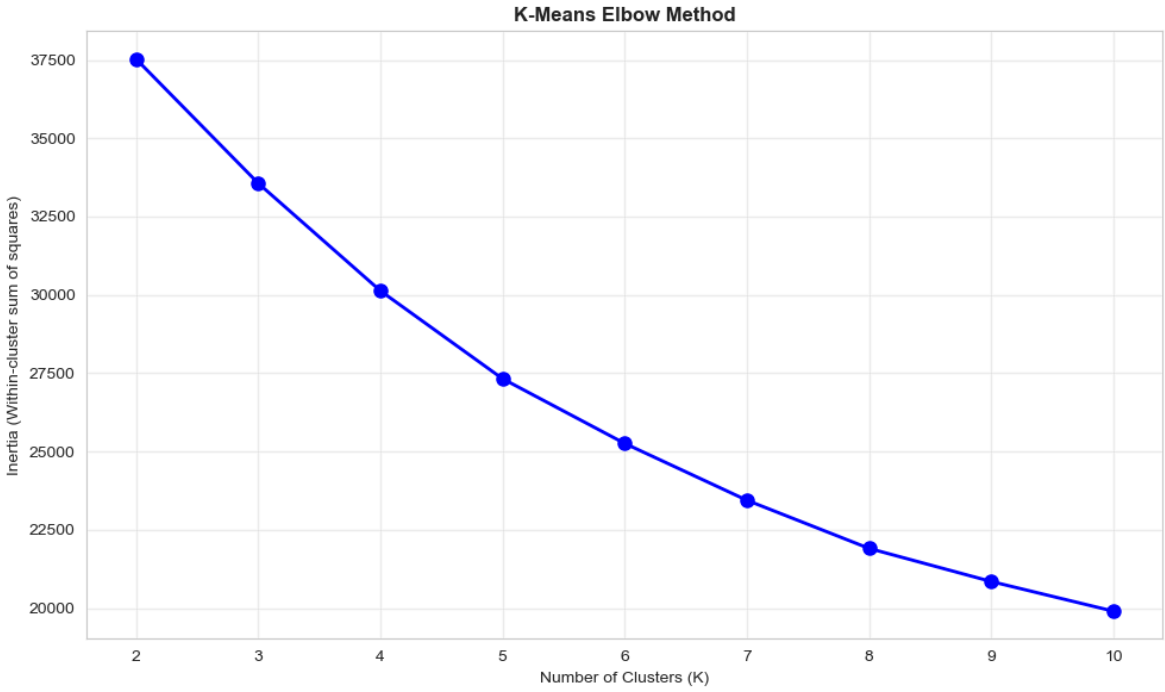
```
# Apply t-SNE (on subset for speed)
import warnings
warnings.filterwarnings('ignore', category=FutureWarning)

subset_size = min(1000, len(data_scaled))
indices = np.random.choice(len(data_scaled), subset_size, replace=False)
data_subset = data_scaled[indices]

tsne = TSNE(n_components=2, random_state=42, perplexity=30, max_iter=1000)  # 改
tsne_result = tsne.fit_transform(data_subset)

# Visualize t-SNE
plt.figure(figsize=(10, 8))
scatter = plt.scatter(tsne_result[:, 0], tsne_result[:, 1],
                      c=df.iloc[indices]['z'], cmap='plasma', alpha=0.6,
                      edgecolors='black', linewidth=0.5)
plt.colorbar(scatter, label='Depth (z)')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.title('t-SNE Visualization (colored by depth)', fontweight='bold')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig(os.path.join(cfg.VIS_DIR, 'tsne_2d.png'), dpi=300, bbox_inches='tigh
plt.show()
```



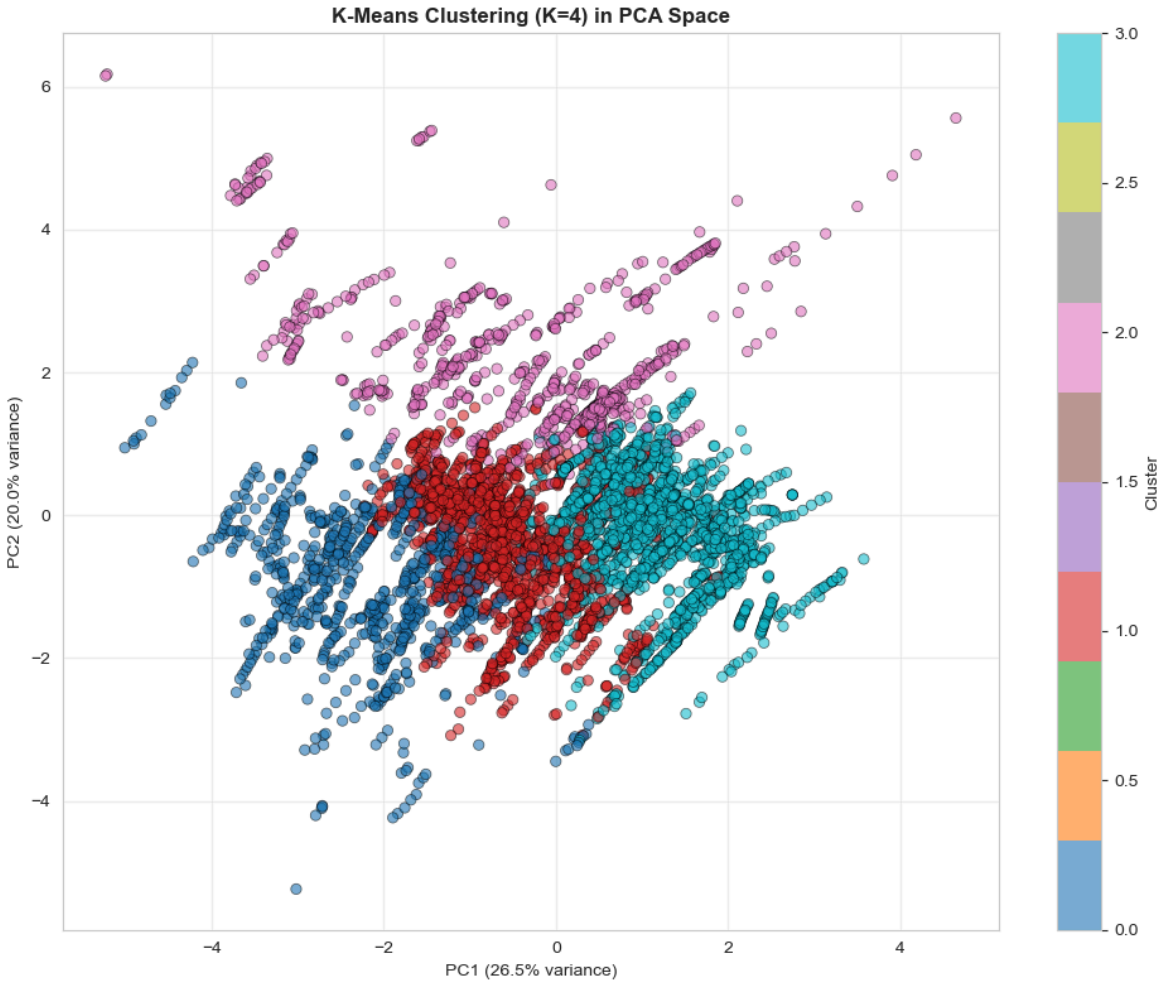
6. Clustering Analysis - K-Means

In [11]:

```
# Elbow method for optimal K
inertias = []
K_range = range(2, 11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(data_scaled)
    inertias.append(kmeans.inertia_)

# Plot elbow curve
plt.figure(figsize=(10, 6))
plt.plot(K_range, inertias, 'bo-', linewidth=2, markersize=8)
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Inertia (Within-cluster sum of squares)')
plt.title('K-Means Elbow Method', fontweight='bold')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig(os.path.join(cfg.VIS_DIR, 'kmeans_elbow.png'), dpi=300, bbox_inches=
plt.show()
```



In [12]:
```python
# Apply K-Means with optimal K
optimal_k = 4
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
clusters = kmeans.fit_predict(data_scaled)

# Visualize clusters in PCA space
plt.figure(figsize=(10, 8))
scatter = plt.scatter(pca_result[:, 0], pca_result[:, 1],
                      c=clusters, cmap='tab10', alpha=0.6,
                      edgecolors='black', linewidth=0.5)
plt.colorbar(scatter, label='Cluster')
plt.xlabel(f'PC1 ({explained_variance[0]:.1%} variance)')
plt.ylabel(f'PC2 ({explained_variance[1]:.1%} variance)')
plt.title(f'K-Means Clustering (K={optimal_k}) in PCA Space', fontweight='bold')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig(os.path.join(cfg.VIS_DIR, 'kmeans_clusters.png'), dpi=300, bbox_inch
plt.show()

# Cluster statistics
df['cluster'] = clusters
print("\n=== Cluster Statistics ===")
print(df.groupby('cluster').mean())
```




```
=== Cluster Statistics ===
                x         y          z    length     width    height  \
cluster
0      -12.222016  2.192860  39.915867  4.039538  1.728502  1.528716
1       -4.235364  1.764063  15.095884  3.935024  1.615433  1.491928
2        1.629682  1.345551  23.114407  4.214312  1.745169  1.690996
3        0.760950  1.682082  26.285326  3.598885  1.623520  1.505510

         rotation_y
cluster
0          1.180338
1          1.645311
2         -0.314619
3         -1.547019
```

7. Clustering Analysis - DBSCAN

In [13]:

```python
# Apply DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_clusters = dbscan.fit_predict(data_scaled)

n_clusters = len(set(dbscan_clusters)) - (1 if -1 in dbscan_clusters else 0)
n_noise = list(dbscan_clusters).count(-1)

print(f"Number of clusters: {n_clusters}")
print(f"Number of noise points: {n_noise}")

# Visualize DBSCAN clusters
plt.figure(figsize=(10, 8))
```

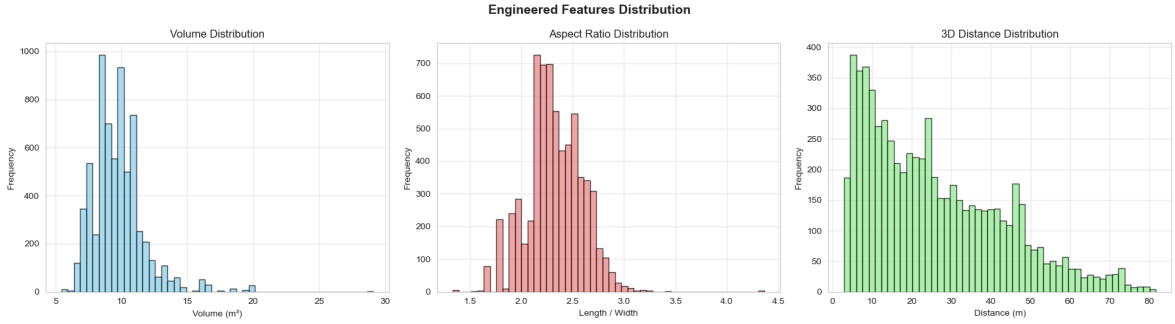
```
scatter = plt.scatter(pca_result[:, 0], pca_result[:, 1],
                      c=dbscan_clusters, cmap='tab10', alpha=0.6,
                      edgecolors='black', linewidth=0.5)
plt.colorbar(scatter, label='Cluster (-1 = noise)')
plt.xlabel(f'PC1 ({explained_variance[0]:.1%} variance)')
plt.ylabel(f'PC2 ({explained_variance[1]:.1%} variance)')
plt.title(f'DBSCAN Clustering in PCA Space', fontweight='bold')
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig(os.path.join(cfg.VIS_DIR, 'dbscan_clusters.png'), dpi=300, bbox_inch
plt.show()
```

```
Number of clusters: 263
Number of noise points: 835
```

8. Feature Engineering Insights

In [15]:
```
# Create additional features
df['volume'] = df['length'] * df['width'] * df['height']
df['aspect_ratio'] = df['length'] / df['width']
df['distance_3d'] = np.sqrt(df['x']**2 + df['y']**2 + df['z']**2)

# Visualize new features
fig, axes = plt.subplots(1, 3, figsize=(18, 5))
fig.suptitle('Engineered Features Distribution', fontsize=14, fontweight='bold')

axes[0].hist(df['volume'], bins=50, edgecolor='black', alpha=0.7, color='skyblue
axes[0].set_title('Volume Distribution')
axes[0].set_xlabel('Volume (m³)')
```

```
axes[0].set_ylabel('Frequency')
axes[0].grid(True, alpha=0.3)

axes[1].hist(df['aspect_ratio'], bins=50, edgecolor='black', alpha=0.7, color='l
axes[1].set_title('Aspect Ratio Distribution')
axes[1].set_xlabel('Length / Width')
axes[1].set_ylabel('Frequency')
axes[1].grid(True, alpha=0.3)

axes[2].hist(df['distance_3d'], bins=50, edgecolor='black', alpha=0.7, color='li
axes[2].set_title('3D Distance Distribution')
axes[2].set_xlabel('Distance (m)')
axes[2].set_ylabel('Frequency')
axes[2].grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig(os.path.join(cfg.VIS_DIR, 'engineered_features.png'), dpi=300, bbox_
plt.show()
```



10. Summary Statistics

In [16]:

```
# Generate comprehensive summary
summary = {
    'Total Samples': len(dataset),
    'Valid Car Objects': len(all_boxes),
    'Average Depth (z)': df['z'].mean(),
    'Average Length': df['length'].mean(),
    'Average Width': df['width'].mean(),
    'Average Height': df['height'].mean(),
    'PCA Components for 95% Variance': np.argmax(cumulative_variance >= 0.95) +
    'Optimal K-Means Clusters': optimal_k,
    'DBSCAN Clusters': n_clusters,
    'DBSCAN Noise Points': n_noise
}

print("\n=== Dataset Summary ===")
for key, value in summary.items():
    if isinstance(value, float):
        print(f"{key}: {value:.2f}")
    else:
        print(f"{key}: {value}")

# Save summary to file
with open(os.path.join(cfg.LOG_DIR, 'data_exploration_summary.txt'), 'w') as f:
    f.write("=== Dataset Exploration Summary ===\n\n")
    for key, value in summary.items():
        if isinstance(value, float):
            f.write(f"{key}: {value:.2f}\n")
```

```
        else:
            f.write(f"{key}: {value}\n")

print("\nSummary saved to:", os.path.join(cfg.LOG_DIR, 'data_exploration_summary
```

```
=== Dataset Summary ===
Total Samples: 7481
Valid Car Objects: 6684
Average Depth (z): 24.496105194091797
Average Length: 3.8390424251556396
Average Width: 1.652369737625122
Average Height: 1.5309635400772095
PCA Components for 95% Variance: 7
Optimal K-Means Clusters: 4
DBSCAN Clusters: 263
DBSCAN Noise Points: 835

Summary saved to: C:\Users\94675\Desktop\group_project_pro\logs\data_exploration_
summary.txt
```