

Discussing CSV-to-Blender with ChatGPT v4

Ron Proctor

ronald.proctor@noirlab.edu

10 April 2024

This document shows a chat log between myself and ChatGPT v4 in March and April 2024. My prompts are shown in [blue](#). My in-line comments are shown in *red italics*.

RP

Are you familiar with the Blender API and what is the most recent version of Blender as of your cutoff date?

Note: This prompt is intended to set the context for the discussion (Blender and the Blender API).

ChatGPT

Yes, I'm familiar with the Blender API. Blender is an open-source 3D creation suite that supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing, and motion tracking, even video editing, and game creation. Blender's API, primarily based on Python, allows for scripting and customization, enabling users to automate tasks, create custom tools, and integrate Blender into their workflows.

As of my last update in April 2023, the most recent stable version of Blender was 3.4. However, Blender is actively developed, and new versions are released regularly. It's possible that a newer version may have been released since my last update. You can check the official Blender website or its repositories for the most current version information.

RP

I have Blender 4.0.2, but I think this should still work: I have a *civ* of x, y, z values. I want to use Blender to display these as particles. Can we achieve this with the Blender API?

Note: My typo in the prompt, "civ" instead of "CSV," was not addressed in the output, nor was it caught or corrected. ChatGPT is often good at picking up details from context, but it missed this one.

ChatGPT

Yes, you can use the Blender API to display a set of x,y,z values as particles. While the specific details can vary depending on the exact version of Blender and the context in which you're working (e.g., creating a new scene, modifying an existing one, types of particles, etc.), the general approach involves creating a mesh object where each vertex represents a point in your dataset, and then using a particle system to display these vertices as particles.

Here's a basic outline of steps you might follow in Blender's Python API to achieve this, keeping in mind you mentioned having Blender 4.0.2:

Prepare Your Data: Ensure your x,y,z values are in a format that can be iterated over (e.g., a list of tuples).

Create a New Mesh: You'll need to create a new mesh object in Blender that will hold your points as vertices.

Add Vertices to the Mesh: Iterate over your dataset and add each point as a vertex to the mesh.

Create a Particle System: Attach a particle system to the mesh object. You can configure this system to represent each vertex as a particle.

Adjust Particle Settings: Depending on your needs, you might want to adjust the size, appearance, and behavior of the particles.

Here is a simplified script that demonstrates this process. Note that this script is meant to be run in Blender's scripting environment and assumes you have basic familiarity with Blender's Python API:

```
import bpy
import bmesh

# Example data: A list of (x, y, z) tuples
data_points = [(1, 1, 1), (2, 2, 2), (3, 3, 3)]

# Create a new mesh and a new object
mesh = bpy.data.meshes.new("ParticleMesh")
obj = bpy.data.objects.new("ParticleObject", mesh)

# Add the object into the scene
scene = bpy.context.scene
scene.collection.objects.link(obj)

# Make the new object active and select it
bpy.context.view_layer.objects.active = obj
obj.select_set(True)

# Create a bmesh, add vertices, and update the mesh
bm = bmesh.new()
for point in data_points:
    bm.verts.new(point)
bm.to_mesh(mesh)
bm.free()
```

```
# Add a particle system to the object
obj.modifiers.new("ParticleSystem", 'PARTICLE_SYSTEM')

# Configure the particle system (basic configuration)
psys = obj.particle_systems[0]
psys.settings.count = len(data_points)
psys.settings.emit_from = 'VERT'
psys.settings.physics_type = 'NO'
psys.settings.use_emit_random = False
psys.settings.particle_size = 0.1 # Adjust size as needed

# Optionally, customize further settings as needed
```

This script is quite basic and intended to get you started. Depending on your project's requirements, you might want to explore more advanced features of the Blender API, such as customizing the particle system further, adding materials to your particles, or integrating this process into a more extensive Blender project workflow. Blender's documentation and community forums can be invaluable resources as you develop more complex implementations.

RP

[My list contains about half a million points. Do you think Blender can handle this?](#)

ChatGPT

Handling half a million points as individual particles in Blender is possible, but it comes with several considerations regarding performance and usability:

Performance: Blender can technically handle large numbers of particles, but the responsiveness and performance of both the viewport and the rendering process will be significantly affected by such a high count. Performance also depends on the specifications of the computer you are using, including the CPU, GPU, and available RAM.

Memory Usage: A high number of particles will increase Blender's memory usage. Ensure your system has enough RAM to manage this, especially if you plan to render the scene or perform complex operations with the particles.

Viewport Usability: With many particles displayed in the viewport, navigating the scene can become sluggish. Blender offers different viewport shading and display modes to help mitigate this, such as bounding box or wireframe modes, which can improve interactivity.

Optimization Techniques:

Level of Detail (LOD): For visualization purposes, consider using lower levels of detail for particles that are farther from the camera.

Particle Instancing: Instead of creating unique geometry for each particle, you can instance a single object across many particle positions to save memory and improve performance.

Chunking: If the dataset allows, consider splitting it into smaller chunks and managing these separately, which can help in both editing and rendering phases.

Use of Particle Systems: For very dense datasets, consider whether a mesh with a vertex for each point might be more efficient than a particle system, especially if you do not need the dynamics that particle systems offer.

Rendering: For rendering, consider using cycles with GPU rendering if available, as it can handle large numbers of particles more efficiently than the CPU in some cases. Also, explore simplifying the scene or using render layers to manage complexity.

If you're encountering performance issues, it may be worthwhile to assess what level of detail is genuinely necessary for your project. Sometimes, not all data points need to be visualized to achieve the desired outcome, or alternative visualization methods can be considered that are more performant.

In summary, while Blender is capable of handling large datasets, doing so efficiently requires careful planning and possibly adapting your approach based on the performance characteristics of your system and the specific needs of your project.

RP

Would it be simpler to not use a particle system and find another way to display the vertices as points?

ChatGPT

Yes, in some cases, especially when dealing with very large datasets like half a million points, it might be simpler and more efficient to display the vertices as points without using a particle system. Blender's particle system offers a lot of flexibility and features for animating and customizing the appearance of particles, but it can be overkill for merely visualizing static points. Additionally, the overhead of managing a particle system can significantly impact performance for large datasets.

An alternative approach to visualize such a large number of points is to use a mesh with vertices representing each point. You can then use a material to make these vertices visible in the render. Here's a simplified approach:

Create a Mesh with Vertices for Each Point: Just as with the particle system approach, you would create a mesh where each point in your dataset is represented by a vertex in the mesh. This has already been covered in the previous example script.

Use a Vertex Shader to Display the Points: Instead of relying on particles, you could use Blender's shading system to create a material that visually emphasizes the vertices. For instance, you can create a shader that uses the geometry node to display each vertex as a point or small sphere. This approach is more lightweight than a particle system and can be more efficient for rendering static datasets.

Optimize the Viewport Display: For navigating the scene in the viewport, you might not be able to see all vertices clearly, especially if they are very dense. Blender's viewport shading options, like "Wireframe" or "Solid" mode, can help improve performance, but they might not display each vertex as a distinct point. Custom OpenGL drawing scripts could be used for a more optimized viewport display, but this requires more advanced scripting.

Here's a basic concept of how you might set up a mesh with a custom shader to visualize vertices:

After creating your mesh and adding vertices as described in the previous examples, create a new material for the object.

In the Shader Editor, use the "Emission" shader node to create a simple shader that will make the vertices visible when rendered. Blender's render engines (Cycles or Eevee) do not render vertices directly, so this approach is more about creating a visual effect that approximates the appearance of points.

For actual point visualization without geometry rendering in Cycles or Eevee, you'd typically need to represent each point with some form of geometry, such as a very small sphere or cube, but this can quickly become impractical for large datasets due to the increased render times and memory usage.

Given these limitations, for purely scientific visualization purposes where performance and direct visualization of large datasets are priorities, external tools specialized in scientific data visualization or custom OpenGL/DirectX applications might be more appropriate. Blender excels in 3D modeling, animation, and rendering, but it's not primarily designed for handling extremely large datasets as efficiently as some specialized tools.

RP

Okay. How about a different approach: Can we convert the css to point cloud data (pcd) and then visualize that with Blender?

ChatGPT

Yes, converting your data to a point cloud data (PCD) format and then visualizing it in Blender is a viable approach. PCD is a file format commonly used for storing point cloud data, often used in the context of 3D scanning and computer vision applications. Blender does not natively support PCD files, but you can use third-party addons or scripts to import PCD files into Blender. Here's how you might approach this:

Step 1: Convert Your Data to PCD

Before importing your data into Blender, you'll need to convert your x,y,z coordinates into the PCD format. This typically involves creating a file that adheres to the PCD structure, including a header specifying the fields (e.g., x, y, z) and the data points themselves.

If you're working with a programming language like Python, you can write a simple script to convert your data into a PCD file. Here's a very basic example of how such a script might look:

```
def write_pcd(xyz_data, filename):
    with open(filename, 'w') as file:
        # PCD header
        file.write("VERSION .7\n")
        file.write("FIELDS x y z\n")
        file.write("SIZE 4 4 4\n")
        file.write("TYPE F F F\n")
        file.write("COUNT 1 1 1\n")
        file.write(f"WIDTH {len(xyz_data)}\n")
        file.write("HEIGHT 1\n")
        file.write("VIEWPOINT 0 0 0 1 0 0 0\n")
        file.write(f"POINTS {len(xyz_data)}\n")
        file.write("DATA ascii\n")

        # Point data
        for point in xyz_data:
            file.write(f"{point[0]} {point[1]} {point[2]}\n")

# Example: Convert and save your data
xyz_data = [(1, 1, 1), (2, 2, 2), (3, 3, 3)] # Replace with
your data
write_pcd(xyz_data, "your_data.pcd")
```

Step 2: Import PCD into Blender

To import the PCD file into Blender, you would typically need an addon or a custom import script, as Blender does not support PCD files by default. As of my last update, there isn't a widely used, official addon for importing PCD files directly into Blender. However, the Blender community often creates scripts and addons for various purposes, including importing different file formats.

You can search online for a Blender PCD import addon or script. If you find one, ensure it's compatible with your version of Blender. If an addon is available, it will typically come with instructions on how to install and use it to import your PCD file.

Alternative Approaches

If you cannot find a suitable add-on for Blender, consider converting your PCD file into a format that Blender supports natively, such as OBJ or STL. These formats are commonly used for 3D models and can be imported into Blender without additional add-ons. There are various tools and libraries available that can convert PCD files to these formats, including Python libraries like `open3d` or `pcl`.

This approach involves an extra conversion step but can be a practical solution if you're unable to import PCD files directly into Blender.

RP

[It looks like Blender supports Point Cache \(pc2\). How would that look?](#)

ChatGPT

Blender's support for Point Cache files (.pc2) offers a more direct path for visualizing point cloud data, although .pc2 files are typically used for storing vertex motion data (such as animations) rather than static point clouds. However, you can use this format to import static point cloud data into Blender by treating each point as a vertex with no motion. Here's how you might approach converting your data to a .pc2 file and then importing it into Blender:

Step 1: Convert Your Data to PC2 Format

To convert your x,y,z data into a Point Cache (.pc2) format, you'll need a script or tool that can write these points into the .pc2 file structure. The .pc2 file format is a binary format that stores information about the count of vertices, frames, and the vertex positions for each frame. Since you're dealing with static point cloud data, you'll only need a single frame.

Writing a .pc2 file from scratch requires understanding the .pc2 binary format, which includes a header followed by the vertex data. The header includes information such as the version of the format, the number of points, and the number of frames.

Step 2: Import PC2 File into Blender

Once you have your .pc2 file, you can import it into Blender using the Mesh Cache modifier. This modifier allows you to apply a point cache file to a mesh object, animating its vertices according to the data in the file. For static point clouds, your mesh will need enough vertices to match the points in your point cloud, and the .pc2 file will position them accordingly.

Here's a simplified step-by-step guide:

Prepare a Mesh Object: In Blender, create a mesh object that has the same number of vertices as your point cloud. A common approach is to create a grid or a highly subdivided plane.

Apply the Mesh Cache Modifier:

Select your mesh object.

Go to the Modifiers tab and add a "Mesh Cache" modifier.

Set the format to "PC2".

Load your .pc2 file by clicking on the folder icon next to "File Path" and navigating to your file.

Since your data is static, ensure the "Play Mode" is set to "Play Once", and adjust the "Frame Start" and "Evaluation Factor" as needed to match your scene's timing.

Adjust as Necessary: You might need to adjust your scene, camera, or object settings depending on how the points are displayed. For static visualization, ensure the animation is set to display the frame where your points are visible.

Creating PC2 Files

Creating a .pc2 file programmatically can be a bit complex due to its binary nature. Unfortunately, I cannot provide a direct script here without knowing more about the specifics of your data and setup. However, the general idea is to write a script that:

Opens a new binary file for writing.

Writes the .pc2 header, specifying the number of vertices and frames (1 frame for a static point cloud).

Iterates through your x,y,z data, writing the vertex positions to the file in the format expected by the .pc2 specification.

Consider searching for existing tools or scripts that can convert to .pc2, or consult the Blender documentation and forums for more detailed guidance on writing .pc2 files. Keep in mind that while using .pc2 for static point clouds in Blender is a bit unconventional, it's a creative approach to visualizing large datasets.

RP

I prefer to keep it scriptable. So, among the options we've discussed, which seems best? Should we consider any other methods?

ChatGPT

Given your preference for a scriptable solution and considering the various options discussed, here are my recommendations:

1. Direct Blender Scripting (Using Mesh Vertices)

Pros: Fully scriptable within Blender; no need for external tools or conversions. Utilizes Blender's native capabilities to handle large datasets efficiently by creating a mesh where each vertex represents a point in your dataset.

Cons: Visualizing each point as a distinct object (like a small sphere) can quickly become impractical for very large datasets due to performance issues. However, using vertices directly avoids this issue, though it means you won't see the points in the viewport as clearly without custom drawing or in render without a specific shader setup.

2. Converting to PCD and Using an Addon or Script for Import

Pros: PCD is a standard format for point cloud data, which might offer compatibility benefits and use outside Blender.

Cons: Requires finding or creating a Blender addon or script for importing PCD, which might be complex and less reliable than using Blender's native features.

3. Using .PC2 Point Cache Files

Pros: Scriptable conversion to .pc2 format allows for using Blender's mesh cache modifier for visualization. This method is somewhat unconventional but can leverage Blender's animation system to display static or animated point clouds.

Cons: Requires a custom script to convert your dataset into the .pc2 format, and the approach is more roundabout, focusing on animation data rather than static visualization.

Best Recommendation: Direct Blender Scripting

Given the considerations, direct Blender scripting to create a mesh with vertices representing your points is likely the best option for your needs. This method offers a good balance between performance, ease of implementation, and scriptability. You can control the entire process within Blender using Python, from generating the mesh based on your data to configuring the viewport and render settings for optimal visualization.

Other Methods to Consider

Custom OpenGL Drawing: For purely scriptable visualization within Blender without creating actual geometry, you could write a custom OpenGL script to draw points in the 3D viewport. This approach requires more advanced knowledge of both OpenGL and Blender's Python API but can be highly efficient for visualizing large datasets.

External Visualization Tools: Depending on your ultimate goal (analysis, presentation, etc.), you might also consider using specialized point cloud or scientific visualization software. Tools like CloudCompare, MeshLab, or even Python libraries like matplotlib (for simpler plots) or PyVista (for 3D visualization) can handle large datasets efficiently and might offer more straightforward scripting opportunities for visualization.

Final Thoughts

The direct Blender scripting approach to create a mesh from your dataset offers a good mix of scriptability, performance, and integration within Blender's ecosystem. However, the best choice ultimately depends on your specific project requirements, including how you plan to use the visualized data, performance considerations, and your familiarity with different tools and scripting languages.

RP

[Please give instructions and code for Direct Blender Scripting with Mesh Vertices.](#)

ChatGPT

For direct Blender scripting with mesh vertices, you'll essentially follow a process to create a new mesh object in Blender, where each vertex corresponds to a point from your dataset. This script will be run inside Blender's scripting environment (Text Editor or Python Console). Here are detailed instructions and example code:

Step 1: Prepare Your Data

Ensure your x,y,z data is in a format that the script can process, typically a list of tuples or a similar iterable structure. For example:

```
data_points = [(x1, y1, z1), (x2, y2, z2), ...]
```

Step 2: Script to Create Mesh from Data Points

This script will create a new mesh object, adding each of your data points as a vertex to the mesh:

```
python
```

```
Copy code
```

```
import bpy
```

```
import bmesh
```

```
# Replace this with your actual data points
```

```
data_points = [(1, 1, 1), (2, 2, 2), (3, 3, 3), ...]
```

```
# Create a new mesh and link it to a new object
```

```
mesh = bpy.data.meshes.new("PointCloudMesh")
```

```
obj = bpy.data.objects.new("PointCloudObject", mesh)
```

```
# Link the object to the scene
```

```
scene = bpy.context.scene
```

```
scene.collection.objects.link(obj)
```

```
# Make the object the active object and ensure it's selected
```

```
bpy.context.view_layer.objects.active = obj
```

```
obj.select_set(True)
```

```
# Create a BMesh, add the vertices, and update the mesh with
the new data
bm = bmesh.new()
for point in data_points:
    bm.verts.new(point)
bm.to_mesh(mesh)
bm.free()
```

Step 3: Run the Script in Blender

Open Blender and switch to the Scripting workspace, which provides a convenient layout for scripting.

Open a new text block in the Text Editor, paste the script above, and adjust your data points as needed.

Run the script by clicking the "Run Script" button in the Text Editor's header, or press Alt + P with the cursor in the Text Editor.

Additional Notes

This script creates a mesh object with vertices at each of your data points' locations. However, these vertices won't be visible in the 3D Viewport by default. They are visible in Edit Mode, or you can use them as emission points for a particle system, or apply a skin modifier, for example, to visualize them in Object Mode.

For very large datasets, Blender's UI might become less responsive. It's often helpful to work in Solid or Wireframe view mode when dealing with large numbers of vertices.

To visualize these points in a render, consider using a particle system where particles are emitted from the vertices, or use another approach to create geometry at each vertex location (e.g., instancing a small sphere at each vertex with a geometry node setup).

This approach directly utilizes Blender's capabilities for 3D data manipulation and visualization, providing a powerful and flexible environment for working with point cloud data or similar datasets.

Let's have the script read the csv file instead of hard coding the input data. We also need to ignore the first line (it is a header).

ChatGPT

To modify the script to read data points from a CSV file and ignore the header line, you'll use Python's built-in csv module. This script assumes that your CSV file is formatted with one point per line, represented by x,y,z coordinates, and that the CSV file is located in a place accessible

by Blender (e.g., in the same directory as your Blender project or somewhere in your system where Blender has read permissions).

Here's how you can adjust the script:

```
import bpy
import bmesh
import csv

# Path to your CSV file
csv_file_path = "/path/to/your/data.csv"

# Function to read points from CSV
def read_points_from_csv(file_path):
    points = []
    with open(file_path, 'r') as file:
        csv_reader = csv.reader(file)
        next(csv_reader) # Skip the header line
        for row in csv_reader:
            # Assuming each row contains x, y, z coordinates as
            # the first three columns
            point = (float(row[0]), float(row[1]),
float(row[2]))
            points.append(point)
    return points

# Read data points
data_points = read_points_from_csv(csv_file_path)

# Create a new mesh and link it to a new object
mesh = bpy.data.meshes.new("PointCloudMesh")
obj = bpy.data.objects.new("PointCloudObject", mesh)

# Link the object to the scene
scene = bpy.context.scene
scene.collection.objects.link(obj)

# Make the object the active object and ensure it's selected
bpy.context.view_layer.objects.active = obj
obj.select_set(True)

# Create a BMesh, add the vertices, and update the mesh with
# the new data
bm = bmesh.new()
for point in data_points:
```

```
        bm.verts.new(point)
    bm.to_mesh(mesh)
    bm.free()
```

Steps to Run the Script

Update the `csv_file_path` variable in the script to point to the actual location of your CSV file.

Open Blender and navigate to the Scripting workspace.

Open a new text block in the Text Editor, copy and paste the modified script, and make sure Blender has access to the specified CSV file path.

Run the script by pressing the "Run Script" button or Alt + P with the cursor in the Text Editor.

This script reads your CSV file, skipping the first line (header), and then proceeds to create a mesh object with vertices corresponding to the x,y,z coordinates from the file. Remember to ensure that the CSV file path is correctly specified and accessible by Blender. If your CSV file is located in a different directory or if you're using Blender on an operating system with specific path conventions (like Windows, which uses backslashes in paths), adjust the path accordingly.

RP

[For csv_file_path, can I read a text object in Blender? How do I do that?](#)

Note: The following code was used to produce graphics and animation for:

<https://noirlab.edu/public/news/noirlab2408/>

ChatGPT

Yes, you can read the contents of a text object (also known as a text block) within Blender's internal text editor using Blender's Python API. This allows you to store your CSV data directly in Blender without needing an external file, which can be especially useful for portability or embedding scripts and data within a single Blender file.

To read data from a text block in Blender, you'll use the `bpy.data.texts` collection. Each item in this collection represents a text block within the Blender file. You can access a specific text block by its name and then read its lines just like you would with a standard file in Python.

Here's how you can adjust the script to read CSV data from a text block instead of an external file:

```
import bpy
import bmesh
import csv
from io import StringIO
```

```

# Name of the text block in Blender
text_block_name = "your_text_block_name_here"

# Function to read points from a text block containing CSV data
def read_points_from_text_block(text_name):
    points = []
    text_block = bpy.data.texts[text_name]
    csv_data = StringIO(text_block.as_string())
    csv_reader = csv.reader(csv_data)
    next(csv_reader) # Skip the header line
    for row in csv_reader:
        # Assuming each row contains x, y, z coordinates as the
        first three columns
        point = (float(row[0]), float(row[1]), float(row[2]))
        points.append(point)
    return points

# Read data points
data_points = read_points_from_text_block(text_block_name)

# Create a new mesh and link it to a new object
mesh = bpy.data.meshes.new("PointCloudMesh")
obj = bpy.data.objects.new("PointCloudObject", mesh)

# Link the object to the scene
scene = bpy.context.scene
scene.collection.objects.link(obj)

# Make the object the active object and ensure it's selected
bpy.context.view_layer.objects.active = obj
obj.select_set(True)

# Create a BMesh, add the vertices, and update the mesh with
the new data
bm = bmesh.new()
for point in data_points:
    bm.verts.new(point)
bm.to_mesh(mesh)
bm.free()

```

Steps to Use the Script

Open Blender and navigate to the Text Editor.

Create a new text block by clicking the "New" button or open an existing one. Make sure the text block contains your CSV data, including a header line.

Name the text block appropriately (you can see and edit the name of the text block at the bottom of the Text Editor window).

Update the `text_block_name` variable in the script to match the name of your text block.

Copy and paste the script into a new or existing text block in the Text Editor and run it.

This approach integrates your CSV data directly into your Blender project, making it easier to manage and share your work without relying on external data files.

RP

Good. Now please modify the code so I can enter a value to skip lines in the CSV. I will use this to plot a subset of the data, which should help performance in Blender while working.

Note: The following code is tested and working.

ChatGPT

To modify the script to allow for skipping a specified number of lines in the CSV data, you can introduce an additional parameter that controls how many lines to skip after the header. This can be useful for working with a subset of your data to improve performance in Blender. Below, I've added a `lines_to_skip` parameter to the function that reads the points. This parameter is used to skip a certain number of lines in the CSV data before starting to read the points.

Here's the updated script with the line-skipping feature:

```
import bpy
import bmesh
import csv
from io import StringIO

# Name of the text block in Blender
text_block_name = "your_text_block_name_here"

# Number of lines to skip after the header before reading data
lines_to_skip = 10 # Adjust this value as needed

# Function to read points from a text block containing CSV
data, with line skipping
def read_points_from_text_block(text_name, skip_lines):
    points = []
    text_block = bpy.data.texts[text_name]
```

```

csv_data = StringIO(text_block.as_string())
csv_reader = csv.reader(csv_data)
next(csv_reader) # Skip the header line

# Skip the specified number of lines
for _ in range(skip_lines):
    next(csv_reader, None)

for row in csv_reader:
    # Assuming each row contains x, y, z coordinates as the
    first three columns
    point = (float(row[0]), float(row[1]), float(row[2]))
    points.append(point)
return points

# Read data points, skipping the specified number of lines
data_points = read_points_from_text_block(text_block_name,
lines_to_skip)

# Create a new mesh and link it to a new object
mesh = bpy.data.meshes.new("PointCloudMesh")
obj = bpy.data.objects.new("PointCloudObject", mesh)

# Link the object to the scene
scene = bpy.context.scene
scene.collection.objects.link(obj)

# Make the object the active object and ensure it's selected
bpy.context.view_layer.objects.active = obj
obj.select_set(True)

# Create a BMesh, add the vertices, and update the mesh with
the new data
bm = bmesh.new()
for point in data_points:
    bm.verts.new(point)
bm.to_mesh(mesh)
bm.free()

```

Instructions for Use

Set the Text Block Name: Update `text_block_name` with the name of your text block that contains the CSV data.

Adjust Lines to Skip: Change `lines_to_skip` to the number of lines you want to skip after the header. This allows you to control the subset of data that gets visualized in Blender, which can help with performance.

Run the Script: Copy and paste the updated script into Blender's Text Editor and run it. Ensure that the specified text block contains your CSV data formatted correctly with a header line followed by data lines.

This updated script provides a flexible way to manage large datasets by allowing you to work with subsets of the data. Adjusting the `lines_to_skip` value lets you quickly change which subset of the data you are visualizing, helping to maintain performance in Blender.

###