# DMPGL 2.0 State Cache Specifications

Version 1.8

PROVISIONAL TRANSLATION

**The content of this document is highly confidential and should be handled accordingly.**

# Table of Contents



# Code



# Tables

# Revision History

| Version | Revision Date | Description |
|---|---|---|
| 1.8 | 2010/06/04 | • Added information on deleting texture states and vertex states. |
| 1.7 | 2010/02/15 | • Renamed “command cache” to “state cache,” both in this document and in its filename. |
| 1.6 | 2009/12/25 | • Mentioned another case in which saving the program state results in an error. |
| 1.5 | 2009/11/30 | • Added an argument used to check the buffer size for each function that saves the state.<br>• Added an argument used to specify the offset of the objects to restore for each function that restores state.<br>• Added specifications for saving data in a format used by the development hardware. |
| 1.4 | 2009/10/30 | • Combined the ***`flag`*** macro names used to restore uniforms for each light source of the program state. |
| 1.3 | 2009/10/02 | • Fixed typos. |
| 1.2 | 2009/09/10 | • Fixed typos. |
| 1.1 | 2009/06/25 | • Explicitly stated that vertex buffer objects include index arrays, as well. |
| 1.0 | 2009/04/30 | • Initial version. |

# 1 State Cache Overview

This document explains the specifications for state cache features in DMPGL 2.0.

The state cache uses fixed units to save and restore each of the setting values and data related to program objects, textures, and vertex data. *State* is the generic name for content that is saved and restored. A *state object* is the unit of data that is saved and restored.

As mentioned earlier, there are three types of state: program state, texture state, and vertex state. Each state is saved and restored used fixed object units. If you configure a state object to be saved, the setting values and data related to that state object will be converted into an internal format and then expanded into a specified memory region. Giving this saved memory region to a state-restoration function causes a state object to be generated for the restored data. You can also overwrite the state of an existing state object with the state to restore.

The settings and data related to each state are managed in fixed groups and can sometimes be saved and restored by group.

# 2 Program State

The program state is saved and restored using program objects generated by **CreateProgram**. In other words, program objects are the state objects.

The following state is stored for each program object.

- The attached shader object and the shader binary data loaded into that shader object
- Uniform settings (for the vertex shader, geometry shader, and fragment shader)
- Settings related to binding attributes

The program state stores all state. You cannot specify which group to save. You can either restore all state or a specified group.

## 2.1 Saving the State

Use the following function to save the state.

**Code 2-1 SaveProgramsDMP**

```
sizei SaveProgramsDMP(
int n, uint* progs, uint flags, sizei bufsize, void* data);
```

The ***progs*** argument specifies a pointer to an array with the program object IDs to be saved.

The ***n*** argument specifies the number of program object IDs stored in ***progs***.

The ***flags*** argument specifies the state group to save. The only supported setting is SAVE_PROGRAMS_DMP, which saves everything. When SAVE_PROGRAMS_CTR_FORMAT_DMP is specified with a bitwise OR on the ***flags*** argument from the PicaOnDesktop (POD) environment, data is saved in a format that can be used on the development hardware. This data cannot be restored from the POD environment. When SAVE_PROGRAMS_CTR_FORMAT_DMP is specified with a bitwise OR on the ***flags*** argument on the development hardware, it is simply ignored.

The ***data*** argument stores the converted data to be saved. This data is required when the program state is restored. If you set the ***data*** argument to 0, data is not saved. Specify the ***bufsize*** argument to be the size (in bytes) of the ***data*** region. An INVALID_OPERATION error occurs if the saved data size is greater than ***bufsize*** when ***data*** is nonzero. In this case, the ***data*** region is not modified.

The return value gives the number of bytes of saved data. An INVALID_VALUE error occurs if the ***flags*** argument is set to an invalid value. An INVALID_OPERATION error occurs if the ***progs*** argument is set to an invalid program object or to a program object that has not been linked properly.

## 2.2 Restoring the State

Use the following function to restore the state.

**Code 2-2 RestoreProgramsDMP**

```
void RestoreProgramsDMP(
int n, uint offset, uint* progs, uint flags, void* data);
```

This restores the program state saved in the ***data*** argument. The program state can be restored in two ways. One method generates a new program object and restores the entire state into it. The other method overwrites an existing program object to restore a partial state. The first method (restoring the full state) is used when the ***flags*** argument is set to `RESTORE_PROGRAMS_DMP` and the second method (restoring a partial state) is used when ***flags*** is set to any other value. When restoring a partial state, you can configure more than one state to be restored using a bitwise OR of setting values. Table 2-1 shows the values that can be set for the ***flags*** argument and the states that are restored.

**Table 2-1 The flags Argument and Restored Program State**

| flags | Restored State |
|---|---|
| `RESTORE_PROGRAMS_DMP` | All state |
| `RESTORE_UPDATE_LIGHTi_PROGRAM_STATE_DMP` (`i` is a light-source ID) | Settings for uniforms whose names include `dmp_FragmentLightSource[i]` (where `i` is a light-source ID) |
| `RESTORE_UPDATE_LIGHTING_PROGRAM_STATE_DMP` | Settings for uniforms whose names include `dmp_FragmentLighting` and `dmp_LightEnv` |
| `RESTORE_UPDATE_MATERIAL_PROGRAM_STATE_DMP` | Settings for uniforms whose names include `dmp_FragmentMaterial` |
| `RESTORE_UPDATE_TEXTURE_BLENDER_PROGRAM_STATE_DMP` | Settings for uniforms whose names include `dmp_TexEnv` |
| `RESTORE_UPDATE_TEXTURE_SAMPLER_PROGRAM_STATE_DMP` | Settings for uniforms whose names include `dmp_Texture[i]` (where `i` is 0, 1, or 2) but not `dmp_Texture[0].perspectiveShadow`, `dmp_Texture[0].shadowZBias`, or `dmp_Texture[0].shadowZScale`. |
| `RESTORE_UPDATE_PROCEDURAL_TEXTURE_PROGRAM_STATE_DMP` | Settings for uniforms whose names include `dmp_Texture[3]` |
| `RESTORE_UPDATE_SHADOW_SAMPLING_PROGRAM_STATE_DMP` | Settings for the uniforms `dmp_Texture[0].perspectiveShadow`, `dmp_Texture[0].shadowZBias`, and `dmp_Texture[0].shadowZScale` |
| `RESTORE_UPDATE_PER_FRAGMENT_OPERATION_PROGRAM_STATE_DMP` | Settings for uniforms whose names include `dmp_FragOperation` |
| `RESTORE_UPDATE_GAS_ACCUMULATION_PROGRAM_STATE_DMP` | Settings for uniforms whose names include `dmp_Gas` |
| `RESTORE_UPDATE_FOG_PROGRAM_STATE_DMP` | Settings for uniforms whose names include `dmp_Fog` |
| `RESTORE_UPDATE_VERTEX_SHADER_STATE_DMP` | Settings for uniforms defined by the vertex shader |

| flags | Restored State |
|---|---|
| RESTORE_UPDATE_GEOMETRY_SHADER_STATE_DMP | Settings for uniforms defined by the geometry shader |

The values RESTORE_UPDATE_VERTEX_SHADER_STATE_DMP and RESTORE_UPDATE_VERTEX_SHADER_STATE_DMP expect the same shader object to be linked to both the program object specified to be saved and the program object specified to be restored. Behavior is undefined if you restore a program object that is linked to a different shader object.

The ***n*** argument specifies the size of the ***progs*** array. When the full state is restored, a new program object for the restored program state is generated and stored in elements with a value of 0 in the array specified by ***progs***. Elements with nonzero values are not affected.

When a partial state is restored, the specified program state is restored for existing programs whose object ID is stored in the array specified by ***progs***. Elements with a value of zero are not affected.

Note that ***progs*** is processed differently depending on the values specified for ***flags***. The program state is restored in ***progs*** in the same order as it was originally saved. The ***offset*** argument specifies the starting index used when the restored program state was originally saved. In other words, the state is restored beginning with the first saved program state when ***offset*** is set to 0 and with the second saved program state when ***offset*** is set to 1.

An INVALID_VALUE error occurs when the sum of ***n*** and ***offset*** exceeds the program state count that can be restored from ***data***. An INVALID_OPERATION error occurs when ***data*** is set to an invalid data region and when ***progs*** is set to an invalid program object. Behavior is undefined when ***flags*** is set to a bitwise OR of RESTORE_PROGRAMS_DMP and other settings.

To use the restored program objects, call **UseProgram**.

# 3 Texture State

The texture state is saved and restored using texture collection objects generated by **GenTextures**. In other words, texture collection objects are the state objects.

All of the texture objects bound to a texture collection, as well as their texture data and configured parameters, are saved for each texture collection object.

You can save and restore the full texture state or you can selectively save and restore particular states.

## 3.1 Saving the State

Use the following function to save the state.

**Code 3-1 SaveTextureCollectionsDMP**

```
sizei SaveTextureCollectionsDMP(
          uint n, uint* txcolls, uint flags, sizei bufsize, void* data);
```

The ***txcolls*** argument specifies a pointer to an array with the texture collection object IDs to be saved.

The ***n*** argument specifies the number of texture collection object IDs stored in ***txcolls***.

The ***flags*** argument specifies the type of state to save. You can specify SAVE_TEXTURE_COLLECTIONS_DMP to save all settings or you can specify a bitwise OR of other setting values. When SAVE_TEXTURE_COLLECTIONS_CTR_FORMAT_DMP is specified from the POD environment using a bitwise OR with SAVE_TEXTURE_COLLECTIONS_DMP or with other setting values, data is saved in a format that can be used on the development hardware. This data cannot be restored from the POD environment. When SAVE_TEXTURE_COLLECTIONS_CTR_FORMAT_DMP is specified as a bitwise OR to the ***flags*** argument on the development hardware, it is simply ignored. Table 3-1 shows the settings that you can specify for ***flags***.

The ***data*** argument stores the converted data to be saved. This data is required when the texture state is restored. If you set the ***data*** argument to 0, data is not saved. Specify the ***bufsize*** argument to be the size (in bytes) of the ***data*** region. An INVALID_OPERATION error occurs if the saved data size is greater than ***bufsize*** when ***data*** is nonzero. In this case, the ***data*** region is not modified.

The return value gives the number of bytes of saved data. An INVALID_VALUE error occurs if the ***txcolls*** argument is set to an invalid value. An INVALID_OPERATION error occurs if an invalid texture object is bound to the texture collection objects specified by ***txcolls***. If ***flags*** is set to a bitwise OR of SAVE_TEXTURE_COLLECTIONS_DMP and other setting values, SAVE_TEXTURE_COLLECTIONS_DMP is ignored and the other setting values are used.

**Table 3-1 The flags Argument and Saved Texture State**

| flags | Saved State |
|---|---|
| SAVE_TEXTURE_COLLECTIONS_DMP | All state |
| SAVE_TEXTURE_COLLECTION_1D_TEXTURES_DMP | Lookup table objects |
| SAVE_TEXTURE_COLLECTION_2D_TEXTURES_DMP | 2D texture objects |
| SAVE_TEXTURE_COLLECTION_CUBE_TEXTURES_DMP | Cube-map texture objects |
| SAVE_TEXTURE_COLLECTIONS_CTR_FORMAT_DMP | Data format for the development hardware |

## 3.2 Restoring the State

Use the following function to restore the state.

**Code 3-2 RestoreTextureCollectionsDMP**

```
void RestoreTextureCollectionsDMP(
          uint n, uint offset, uint* txcolls, uint flags, void* data);
```

This restores the texture state saved in the ***data*** argument.

The ***n*** argument specifies the size of the ***txcolls*** array. A new texture collection object for the restored texture state is created and stored in elements with a value of 0 in the array specified by ***txcolls***. Elements with nonzero values are not affected. Texture collection objects are stored in ***txcolls*** in the same order as they were when the texture state was saved.

The ***offset*** argument specifies the starting index used when the restored texture state was originally saved. In other words, the state is restored beginning with the first saved texture state when ***offset*** is set to 0 and with the second saved texture state when ***offset*** is set to 1.

The ***flags*** argument specifies the type of state to restore. You can specify RESTORE_TEXTURE_COLLECTIONS_DMP, which restores all settings, or a bitwise OR of other setting values, which restores a partial state. If ***flags*** is set to a bitwise OR of RESTORE_TEXTURE_COLLECTIONS_DMP and other setting values, RESTORE_TEXTURE_COLLECTIONS_DMP is ignored and the other setting values are used. Table 3-2 shows the values that can be set for the ***flags*** argument and the states that are restored.

**Table 3-2 The flags Argument and Restored Texture State**

| flags | Restored State |
|---|---|
| RESTORE_TEXTURE_COLLECTIONS_DMP | All state |
| RESTORE_TEXTURE_COLLECTION_1D_TEXTURES_DMP | Lookup table objects |
| RESTORE_TEXTURE_COLLECTION_2D_TEXTURES_DMP | 2D texture objects |
| RESTORE_TEXTURE_COLLECTION_CUBE_TEXTURES_DMP | Cube-map texture objects |

An INVALID_VALUE error occurs when the sum of ***n*** and ***offset*** exceeds the texture state count that can be restored from ***data***. An INVALID_OPERATION error occurs if the ***data*** argument is set to an invalid data region.

To use the restored texture state, call **BindTexture**(TEXTURE_COLLECTION_DMP, ***txcoll***).

Restoring the texture state causes new 2D texture objects, cube-map texture objects, and lookup table objects to be created for the ones that were bound when the state was saved. Data is restored in each of these objects, which are then bound to the texture collection object obtained in ***txcoll***. To delete the recovered texture state, you must individually delete each of the bound texture objects in addition to ***txcoll***. With the texture collection bound by a call to **BindTexture**(TEXTURE_COLLECTION_DMP, ***txcoll***), use **GetIntegerv** to get the IDs of every bound texture object and then call **DeleteTextures** on those IDs. To get the IDs of 2D texture objects, cube-map texture objects, and lookup table objects, call **GetIntegerv** with ***pname*** set equal to TEXTURE_BINDING_2D, TEXTURE_BINDING_CUBE_MAP, and TEXTURE_BINDING_LUTn_DMP, respectively.

# 4 Vertex State

The vertex state is saved and restored using vertex state collection objects generated by **GenBuffers**. In other words, vertex state collection objects are the state objects.

For each vertex state collection object, all of the vertex buffer objects (ARRAY_BUFFER and ELEMENT_ARRAY_BUFFER) bound to the vertex state collection are saved along with the settings from **EnableVertexAttribArray**, **DisableVertexAttribArray**, **VertexAttrib{1234}{fv}**, and **VertexAttribPointer**.

You can save and restore the entire vertex state. You cannot selectively save and restore a particular state.

## 4.1 Saving the State

Use the following function to save the state.

**Code 4-1 SaveVertexStateCollectionsDMP**

```
sizei SaveVertexStateCollectionsDMP(
          uint n, uint* vscolls, uint flags, sizei bufsize, void* data);
```

The ***vscolls*** argument specifies a pointer to an array with the vertex state collection object IDs to be saved.

The ***n*** argument specifies the number of vertex state collection object IDs stored in ***vscolls***.

The ***flags*** argument specifies the type of state to save. The only supported setting is SAVE_VERTEX_STATE_COLLECTIONS_DMP, which saves everything. When SAVE_VERTEX_STATE_COLLECTIONS_CTR_FORMAT_DMP is specified from the POD environment with a bitwise OR on the ***flags*** argument, data is saved in a format that can be used on the development hardware. This data cannot be restored from the POD environment. When SAVE_VERTEX_STATE_COLLECTIONS_CTR_FORMAT_DMP is specified from the development hardware with a bitwise OR on the ***flags*** argument, it is simply ignored.

The ***data*** argument stores the converted data to be saved. This data is required when the vertex state is restored. If you set the ***data*** argument to 0, data is not saved. Specify the ***bufsize*** argument to be the size (in bytes) of the ***data*** region. An INVALID_OPERATION error occurs if the saved data size is greater than ***bufsize*** when ***data*** is nonzero. In this case, the ***data*** region is not modified.

The return value gives the number of bytes of saved data. An INVALID_VALUE error occurs if ***vscolls*** is set to an invalid value. An INVALID_OPERATION error occurs if an invalid vertex buffer object is bound to the vertex state collection objects specified by ***vscolls***.

## 4.2 Restoring the State

Use the following function to restore the state.

**Code 4-2 RestoreVertexStateCollectionsDMP**

```
void RestoreVertexStateCollectionsDMP(
          uint n, uint offset, uint* vscolls, uint flags, void* data);
```

This restores the vertex state saved in the ***data*** argument.

The ***n*** argument specifies the size of the ***vscolls*** array. A new vertex state collection object for the restored vertex state is created and stored in elements with a value of 0 in the array specified by ***vscolls***. Elements with nonzero values are not affected. Vertex state collection objects are stored in ***vscolls*** in the same order as they were when the vertex state was saved.

The ***offset*** argument specifies the starting index used when the restored vertex state was originally saved. In other words, the state is restored beginning with the first saved vertex state when ***offset*** is set to 0 and with the second saved vertex state when ***offset*** is set to 1.

The ***flags*** argument specifies the type of state to restore. The only supported setting is RESTORE_VERTEX_STATE_COLLECTIONS_DMP, which restores everything.

An INVALID_VALUE error occurs when the sum of ***n*** and ***offset*** exceeds the vertex state count that can be restored from ***data***. An INVALID_OPERATION error occurs if the ***data*** argument is set to an invalid data region.

To use the restored vertex state, call **BindBuffer**(VERTEX_STATE_COLLECTION_DMP, ***vscoll***).

Restoring the vertex state causes a new vertex buffer object to be created for every one that was bound when the state was saved. Data is restored in each of these objects, which are then bound to the vertex state collection obtained in ***vscoll***. To delete the recovered vertex state, you must individually delete each of the bound vertex buffer objects in addition to ***vscoll***. With the vertex state collection bound by a call to **BindBuffer**(VERTEX_STATE_COLLECTION_DMP, ***vscoll***), use **GetIntegerv** and **GetVertexAttribiv** to get the IDs of every bound vertex buffer object and then call **DeleteBuffers** on those IDs. Call **GetIntegerv** with ***pname*** set equal to ARRAY_BUFFER_BINDING and ELEMENT_ARRAY_BUFFER_BINDING to get the IDs of every bound vertex buffer object. Call **GetVertexAttribiv** with ***pname*** set equal to VERTEX_ATTRIB_ARRAY_BUFFER_BINDING to get the IDs of vertex buffer objects bound to a vertex array.