

STL Texture replacement PBR

För STL Script Library v4.64.0

För vagnar som är byggda med den moderna PBR-tekniken fungerar inte riktigt de gamla scripten **STLVehicle.gs** och **STLMotorVehicle.gs**, eftersom det bara kan byta ut en textur per vagn. Med PBR-texturer använder man flera stycken *PBR-maps*, som i Trainz är packade till tre mappar och kallas för albedo, normal och parameter. För att kunna göra texture-replacement med sådana, "tredelade" texturer så finns det nu nya script **STLVehiclePBR.gs** och **STLMotorVehiclePBR.gs** som kompletterar de gamla scripten.

Dessa nya script **STLVehiclePBR.gs** och **STLMotorVehiclePBR.gs** finns med i STL Script Library version v4.64.0 och uppåt, vilken har kuid-version 5 (alltså kuid <kuid2:177292:209000:5>) eller högre.

Scripten gör det möjligt att i Surveyor öppna propertyrutan för fordonet och därifrån ställa in vilken PBR-textur man vill ha från flera olika.

Detta dokument kommer använda begreppet "textur" för en hel PBR-textur bestående av albedo, normal och parameter, samt begreppet "map" för varje enskild komponent (albedo, normal eller parameter) av en PBR-texture.

Både de gamla **STLVehicle.gs/STLMotorVehicle.gs** och de nya **STLVehiclePBR.gs/STLMotorVehiclePBR.gs** ärver funktionalitet från scripten **MsVehicle.gs/MsMotorVehicle.gs**, vilka är de script som implementerar STL-koppel-funktionaliteten. Med andra ord får man automatiskt STL-koppel-funktionaliteten om man använder antingen **STLVehicle** eller **STLVehiclePBR** i sin vagn eller om man använder **STLMotorVehicle.gs** eller **STLMotorVehiclePBR.gs** i sitt motordrivna fordon.

Använd följande punkter för att bestämma vilken av de tre scripten **MsVehicle**, **STLVehicle** och **STLVehiclePBR** som din vagn ska ärva för att få den funktionalitet du önskar:

- Om du har en vagn med STL-koppel, men där du *inte* vill kunna byta textur: använd **MsVehicle**. Detta gäller både för fordon med PBR- och gamla icke-PBR-texturer.
- Om du har en vagn med STL-koppel och PBR-texture där du vill kunna byta textur: använd **STLVehiclePBR**.
- Om du har en vagn med STL-koppel och den gamla typen av icke-PBR-texturer där det ska vara möjligt att byta textur: använd **STLVehicle**.

Motsvarande förhållande gäller för motorfordon med **MsMotorVehicle/STLMotorVehicle/STLMotorVehiclePBR**. Resten av detta dokument kommer att hänvisa enbart till **STLVehiclePBR**, men samma saker gäller för **STLMotorVehiclePBR** om ej annat anges.

En skillnad mellan det gamla **STLVehicle** och det nya **STLVehiclePBR** är att det nya scriptet inte har dörrfunktionaliteten som fanns i **STLVehicle**. Önskas sådan funktionalitet så behöver det implementeras separat, per vagn, vilket också gör det lättare att anpassa det beroende på vagnstyp. Däremot har det nya **STLVehiclePBR** en hjälpmetod som gör det enkelt att lägga till egen text i propertyrutan i Driver-mode, likt hur man kan lägga till HTML i propertyrutan i Surveyor med hjälp av

```
string GetDescriptionHTML(void)
```

Den nya funktionen har signaturen:

```
string GetDriverDescriptionHTML(string html)
```

Till skillnad från `GetDescriptionHTML` tar `GetDriverDescriptionHTML` in en sträng html, vilket är HTML sidan, så långt som den har byggts fram tills dess att de egna tilläggen kan adderas.

Sammanfattningsvis finns dessa funktioner i de tre scripten:

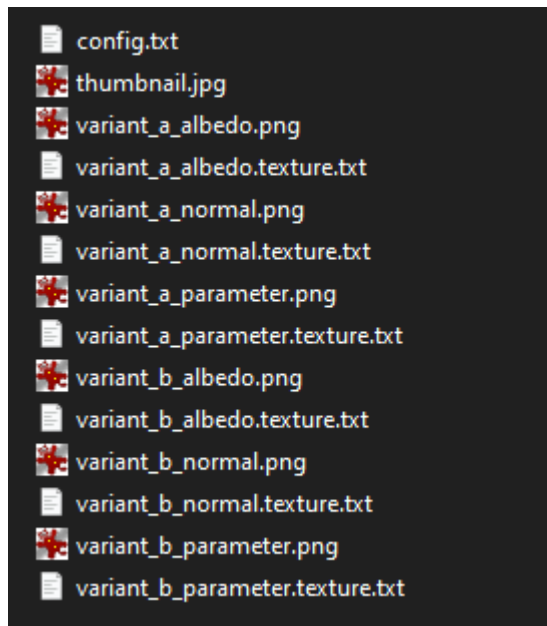
Script\Funktionalitet	STL-koppel	Texture replacement	Dörrscript	GetDriverDescriptionHTML
MsVehicle	Ja	Nej	Nej	Nej
STLVehicle	Ja	Ja, enkla texturer	Ja	Nej
STLVehiclePBR	Ja	Ja, med PBR	Nej	Ja

För användning av de gamla scripten **STLVehicle.gs** och **STLMotorVehicle.gs** hänvisas till den gamla manualen.

Att använda STLVehiclePBR

Skapa först en ny texture-group-asset där textur-varianterna kommer att sparas. Din grundtextur med dess tre PBR-mappar kommer att ligga i vagnens asset, precis som förut, men de extra textur-varianterna kommer att ligga i texture-groupen.

Exemplet nedan visar hur det fungerar för en vagn med tre texturer. Då ligger den första texturen med dess tre PBR-mappar i vagnens asset och de andra två varianterna med sina totalt 6 PBR-mappar i texture-groupen. Innehållet i en sådan texture-group borde se ut något i stil med följande:



Här kallas de två PBR-texture-varianterna för variant_a och variant_b, men du kan naturligtvis ha vilka namn du vill på dem. Varje **variant_x_y.texture.txt** pekar ut motsvarande png-fil, precis som motsvarande filer för default-texturen i vagnens asset. Notera att Trainz inte genererar texture.txt-filerna automatiskt när du committar assets i Content Manager, som det gör för vagnens asset, utan dem behöver man skapa själv, eller kopiera och modifiera från default-texturen i vagnens asset.

Texture-groupen:s config för detta exempel skulle se ut så här:

```
trainz-build          4.6
category-class        "JO"
category-region       "SE"
category-era          "1900s"
username              "Ett namn på asseten"
kind                  "texture-group"

textures
{
  0                    "variant_a_albedo.texture"
  1                    "variant_a_normal.texture"
  2                    "variant_a_parameter.texture"
```

```

3                "variant_b_albedo.texture"
4                "variant_b_normal.texture"
5                "variant_b_parameter.texture"
}

string-table
{
    texture_0     "Grundtextur"
    texture_1     "Vädrad"
    texture_2     "Nymålad"
}
kuid             <kuid...>

thumbnails
{
    0
    {
        image     "thumbnail.jpg"
        width     480
        height    360
    }
}

kuid-table
{
}

```

I configen ska de olika mapparna som finns i texture-group:en pekats ut i texture-blocket. Här pekas bara de texturer som ligger i texture-groupens asset; default-texturen i vagnens asset kommer att hittas automatiskt. Här är det viktigt med ordningen. PBR-mapparna ska listas för varje PBR-textur-variant i ordningen albedo, normal, parameter. Så i vårt exempel: albedo, normal och parameter för variant A, sedan albedo, normal och parameter för variant B, osv.

Därefter lägger man till namnet som ska visas i propertyrutan på alla textur-varianterna i string-table:n. Här inkluderar man grundtexturen som ligger i vagnens asset, vilket är den första texturen som indexeras med 0. Så för vårt exempel mappas de på följande sätt:

- texture_0: Grundtextur
- texture_1: Variant A
- texture_2: Variant B

Detta är allt som krävs i texture-group:en. Här behövs inget script, det finns istället i vagnens asset.

Tillägg i fordonets asset

I fordonets asset behöver man lägga till scriptet, peka ut texture-group:en, samt peka ut vilka meshar och vilka textur-mappar på vagnen som ska kunna bytas ut.

Se först till att lägga till STLs scriptbibliotek i kuid-table för vagnen, samt lägg till det i blocket **script-include-table** för att Trainz ska veta att den ska leta efter script i den asseten. Notera att vi inkluderar kuid-version 5 av asseten här, då det är i den versionen av STL Script Library som **STLVehiclePBR**-scriptet lades till. Vi lägger även till texture-group:en i kuid-tablen. Denna måste ha namnet **pbr_texture_asset** för att scriptet ska hitta den.

```
...
script-include-table
{
    mscommonsource                <kuid2:177292:209000:5>
}

kuid-table
{
    pbr_texture_asset             <kuid...>
    mscommonsource                <kuid2:177292:209000:5>
    ...
}
```

Därefter behöver vagnen ett eget script, om den inte redan har det. Finns det redan ett script, se bara till att det inkluderar och ärver **STLVehiclePBR**. Annars, lägg till ett nytt script som innehåller följande kod:

```
include "STLVehiclePBR.gs"

class ThisVehicle isclass STLVehiclePBR {
};
```

Det skapar en tom klass som ärver all funktionalitet från STLVehiclePBR. Klassens namn **ThisVehicle** kan bytas ut mot ett annat namn som bättre matchar vagnens, om så önskas. Döp filen till samma som klassen, men med bara små bokstäver: **thisvehicle.gs**

I config-filen, peka ut scriptet och klassen, om det inte redan är gjort:

```
...
script                "thisvehicle.gs"
class                 "ThisVehicle"
...
```

Till sist behöver man också peka ut vilka mesher i mesh-tablen som ska ha texturer-replacement, samt vilka texture-mappar i den meshen som ska kunna bytas ut. Här behöver man peka ut alla tre PBR-mapparna, på detta sättet:

```

mesh-table
{
  default
  {
    mesh                                "main_mesh.lm"
    auto-create                          1

    effects
    {
      textures_0_albedo
      {
        kind                            "texture-replacement"
        texture                          "defaulttexture_albedo.texture"
      }

      textures_0_normal
      {
        kind                            "texture-replacement"
        texture                          "defaulttexture_normal.texture"
      }

      textures_0_parameter
      {
        kind                            "texture-replacement"
        texture                          "defaulttexture_parameter.texture"
      }
    }
  }

  door_left
  {
    mesh                                "door_left.lm"
    auto-create                          1

    effects
    {
      textures_1_albedo
      {
        kind                            "texture-replacement"
        texture                          "defaulttexture_albedo.texture"
      }

      textures_1_normal
      {
        kind                            "texture-replacement"
        texture                          "defaulttexture_normal.texture"
      }

      textures_1_parameter
      {
        kind                            "texture-replacement"
        texture                          "defaulttexture_parameter.texture"
      }
    }
  }
  ...
}

```

I detta exempel har meshens default-texturer texture-filer som heter **defaulttexture_X.texture.txt**, där X är albedo, normal eller parameter för de olika mapparna. Dessa namn kan vara vad som helst egentligen.

Varje mesh som ska kunna byta texturer måste ha tre effekter av typen **texture-replacement** som ska heta **textures_X_albedo**, **textures_X_normal** och **textures_X_parameter**, där X är ett löpande nummer och varje effekt pekar på den texture-map den ska byta ut. Här är namnet viktigt för att scriptet letar efter effekter med dessa specifika namn. Varje mesh som man byter ut texturer på måste ha ett unikt namn på effekten, så därför finns siffran med i namnet. Bara öka denna siffra löpande för varje mesh. Scriptet har för närvarande stöd för upp till 20 separata mesher med texture replacement, med namn **textures_0_albedo/textures_0_normal/textures_0_parameter** till **textures_19_albedo/textures_19_normal/textures_19_parameter**.

I det gamla scriptet fanns det 11 namn med olika förkortningar som textures_dr, där dr stod för "door right" till exempel, men i PBR-scriptet är detta utbytt mot löpande siffror, eftersom de inte på något sätt är knutna till vad meshen används till.

Notera att scriptet stannar när den inte hittar fler effekter, så om configen har **textures_0_X**, **textures_1_X** och sedan ett hopp till **textures_3_X**, så kommer scriptet att stanna efter de första två då den inte hittar **textures_2_X**.

Att lägga till nya texturer

Om du i efterhand vill lägga till nya texturer så behöver du bara ändra i texture-gruppen. Lägg bara till de tre png-filerna, skapa .texture.txt-filer som pekar ut rätt png-fil och lägg till referenser till dessa sist i **texture**-blocket i configen. Lägg sedan till ett nytt namn på texturen sist i string-table:n. Spara asseten, så ska allt fungera. Vagnens script läser automatiskt in alla texturer från texture-gruppen, utan att några ändringar i vagnens asset behöver göras.

Använda samma PBR-map till flera texturer

I vissa fall använder olika textur-variationer samma PBR-mappar som en annan textur-variation. Detta gäller nog framförallt normal och parameter-texturerna. Så till exempel, om din texture-group innehåller flera PBR-texturer som använder samma normal-map, behöver du inte ha flera kopior av normal-mappen med olika namn i asseten som tar upp onödig plats, utan kan peka på samma .texture-fil i texture-blocket i config-filen.

```
...

textures
{
    0                "variant_a_albedo.texture"
    1                "default_normal.texture"
    2                "default_parameter.texture"
    3                "variant_b_albedo.texture"
    4                "default_normal.texture"
    5                "default_parameter.texture"
}

...
```

Om defaulttexturen i vagnens asset har samma textur som varianterna i texture-group:en behöver man dock ha en kopia av denna map i texturegroupens asset, då man inte kan referera till map:en i vagnens asset ifrån texture-blocket i texture-group:en.

Vagnar med flera texturer

Scriptet **STLVehiclePBR** har enbart stöd för att byta ut en PBR-textur per vagn. För att kunna byta flera texturer måste man göra ett eget script som antingen implementerar all den logiken själv, eller bygger vidare på **STLVehiclePBR**. Dock går det bra att använda **STLVehiclePBR** på vagnar med mesher som består av flera PBR-texturer, men där bara en ska bytas. Exempelvis om man har en stor PBR-textur för större delen av vagnen och en liten PBR-textur med bara nummerskyltarna, då kan man använda **STLVehiclePBR** för att kunna byta vagnsnummer-texturen, men alltid behålla samma textur med värderingen på själva vagnen.

Att använda `GetDriverDescriptionHTML`

Trainz tillhandahåller en smidig funktion `GetDescriptionHTML` som möjliggör att lägga till egen text med hjälp av enkel HTML i propertyrutan i Surveyor mode, alltså den rutan man får upp om man väljer frågetecken-knappen och sedan trycker på fordonet. Dock finns inget lika enkelt sätt att lägga till egen HTML-kod i motsvarande propertyruta i Driver mode, som man får upp om man högerklickar på fordonet och väljer View Details. **STLVehiclePBR** tillhandahåller därför en sådan metod som kan override:as av de script som använder **STLVehiclePBR** för att kunna lägga till text och länkar för att göra saker som att kunna öppna dörrar eller andra typer av luckor på fordonet i spelet. Detta kan göras på följande sätt:

```
...

string GetDriverDescriptionHTML(string html) {
    html = html + "<p>Lite HTML-kod som du vill lägga till</p><br>";
    ...
    return html;
}

...
```