Práctica 6:

Texturas

ÍNDICE

ÍNDICE **2**

1. Manual **3**

2. Diseño y Justificación de Clases **4**

2.1. PagSubdivisionProfile *4*

2.2. PagRevolutionObject *4*

2.3. Pag3DElement *5*

2.4. Pag3DGroup *5*

2.5. PagPlane *5*

2.6. PagVAO *6*

2.7. PagCamera *6*

2.8. PagLightSource *7*

2.9. PagLightAplication *7*

2.10. PagAmbientLight *8*

2.11. PagSpotLight *8*

2.12. PagPuntualLight *9*

2.13. PagMaterial *9*

2.14. PagLibraryMaterial *9*

2.15. PagTexture *10*

2.16. PagLibraryTexture *10*

2.17. PagRenderer *11*

3. Capturas escena final y de procedimiento **12**

1. MANUAL

Asignación de teclas para el control de la aplicación:

P: activación dibujo de puntos

W: activación dibujo de líneas/alambres

M: activación dibujo de malla de triángulos

N: activación modo debug normales

T: activación modo debug tangentes

C: activación modo debug coordenadas de textura

L: todas las luces de la escena

A: sólo luz ambiental

D: sólo luz puntual

F: sólo luz focal

X: Escena final iluminada y con texturas

0: movimiento stop

1: movimiento orbit

2: movimiento pan

3: movimiento truck

4: movimiento crane

5: movimiento dolly

6: cambio de cámara (frontal y aérea)

Rueda: zoom

Esc: cerrar la ventana

2. DISEÑO Y JUSTIFICACIÓN DE CLASES

En este apartado se explicará el diseño, atributos y métodos de las clases principales de la aplicación gráfica.

2.1. PagSubdivisionProfile

Clase encargada del perfil de subdivisión.

Tiene los siguientes atributos; **vPoints** (vector que almacena los puntos de perfil), **isValidProfile** (booleano que indica si el perfil es válido), **hasBottomFan** (booleano que indica si tiene tapa inferior) y **hasTopFan** (booleano que indica si tiene tapa superior).

Sus métodos son:

Privados: **subdivision,** método encargado de ejecutar el algoritmo de subdivisión sobre el vector de puntos, esta función será llamada por el *subdivide*.

Públicos: **subdivide**, ejecuta la función subdivide *times* veces y devuelve el nuevo perfil, **getVPoints**, que devuelve el vector de puntos, **getIsValidProfile**, que devuelve si el perfil es válido, **getHasBottomFan**, que devuelve si el perfil tiene tapa inferior y **getHasTopFan**, que devuelve si el perfil tiene tapa superior.

2.2. PagRevolutionObject

Clase encargada de los objetos de revolución.

Tiene los siguientes atributos; **profile** (perfil que será revolucionado para formar el objeto, será de tipo *PagSubdivisionProfile*), **body** (PagVAO correspondiente al cuerpo del objeto), **bottomFan** (PagVAO correspondiente a la tapa inferior del objeto) y **topFan** (PagVAO correspondiente a la tapa superior del objeto).

Sus métodos son:

Públicos: **isValid**, devuelve si el perfil es válido, **has**, que devuelve si el objeto tiene la parte que se le pase por parámetros, **getNPoints**, que devuelve el número de puntos del objeto, **drawAsPoints** / **drawAsLines** / **drawAsTriangles**, que dibujan el objeto como una

nube de puntos, un conjunto de líneas/alambres y como malla de triángulos (se les pasará un shaderProgram, el ads, ya que en él se aplicará el material y las luces), en ellas se llamarán a los métodos de cada *PagVAO*.

2.3. Pag3DElement

Clase abstracta encargada de los objetos 3D. Esta será la clase padre de la que heredarán *PagRevolutionObject*, *Pag3DGroup* y *PagPlane*.

Tiene los siguientes atributos; **modelMatrix** (matriz de modelado del objeto) y **material** (material que será aplicado al objeto).

Sus métodos son:

Públicos: **getMaterial y setMaterial**, devuelve el material asignado y lo modifica, **drawAsPoints** / **drawAsLines** / **drawAsTriangles**, métodos abstractos que serán implementados en las clases hijas y **translate** / **rotate** / **scale**, que permiten trasladar, rotar y escalar un objeto 3D.

2.4. Pag3DGroup

Clase encargada de agrupar objetos *Pag3DElement*.

Tiene los siguientes atributos; **elements** (vector de *Pag3DElement* que almacena los elementos 3D pertenecientes al grupo).

Sus métodos son:

Públicos: **insertElement**, inserta un elemento 3D en el vector y **drawAsPoints** / **drawAsLines** / **drawAsTriangles**, que dibujan cada objeto del vector como una nube de puntos, un conjunto de líneas/alambres y como malla de triángulos.

2.5. PagPlane

Clase encargada de los planos.

Tiene los siguientes atributos; **plane** (PagVAO correspondiente al plano), **width** (anchura del plano), **height** (altura del plano), **horizontalSquares** (casillas horizontales del plano) y **verticalSquares** (casillas verticales del plano).

Sus **métodos** son:

Públicos: **drawAsPoints** / **drawAsLines** / **drawAsTriangles**, que dibujan el plano como una nube de puntos, un conjunto de líneas/alambres y como malla de triángulos.

2.6. PagVAO

Clase encargada de los vertex arrays object.

Tiene los siguientes **atributos**; **vPosNorm** (vector que almacena las posiciones y normales del objeto), **vCoordText** (vector que almacena las coordenadas de textura del objeto), **vTang** (vector que almacena las tangentes del objeto), **vPointsIndex** (vector que almacena los índices de los puntos del objeto), **vLinesIndex** (vector que almacena los índices de las líneas/alambres del objeto), **vao** (contiene los vbo y los ibo), **vbo** (vector de 3 elementos, en la posición 0 -> vPosNorm, en la posición 1 -> vCoordText y en la posición 2 -> vTang), **ibo** (vector de 3 elementos, en la posición 0 -> vTrianglesIndex, en la posición 1 -> vPointsIndex y en la posición 2 -> vLinesIndex).

Sus **métodos** son:

Públicos: **inserts de cada vector**, permiten insertar los elementos a cada uno de los vectores que existen como atributos, **size**, que devuelve el número de puntos del objeto, **drawAsPoints** / **drawAsLines** / **drawAsTriangles**, que dibujan el objeto como una nube de puntos, un conjunto de líneas/alambres y una malla de triángulos, y **fillGeometry**, será llamado una vez se inserten todos los elementos en los vectores con el fin de rellenar la geometría de estos.

2.7. PagCamera

Clase encargada de la cámara de la escena.

Tiene los siguientes **atributos**; **viewMatrix** (matriz de visión), **projMatrix** (matriz de proyección), **position** (posición de la cámara), **lookAt** (punto al que mira la cámara), **up** (vector arriba), **fovY** (ángulo de visión en y), **zNear** (marca el límite de recorte de la escena más cercano), **zFar** (marca el límite de recorte de la escena más lejano), **width** (ancho de la ventana), **height** (alto de la ventana), v (parámetro user-friendly v), u

(parámetro user-friendly u), n(parámetro user-friendly n), viewMatrix (matriz de visión) y projMatrix (matriz de proyección).

Sus **métodos** son:

Privados: **calculateV**, encargado de calcular el parámetro user-friendly v cada vez que se mueve la cámara, **calculateViewMatrix** y **calculateProjMatrix**, encargados de calcular la matriz de visión y proyección respectivamente (usando *glm::lookAt* y *glm::perspective*). Estos métodos han sido implementados para no duplicar código, ya que esta operación se calcula de manera recurrente.

Públicos: **setCamera**, que permite cambiar todos los parámetros de la cámara en una única llamada, **getters de la matriz** de visión, de la matriz de proyección y de la matriz de visión proyección (que se calcula con el producto de las dos anteriores), **movimientos** de cámara (concretamente orbit, pan, truck, crane, dolly y zoom) y **setWidth**/**setHeight**, que permiten modificar el tamaño de la ventana y que serán usados para que el objeto no pierda sus dimensiones al redimensionar la ventana.

2.8. PagLightSource

Clase encargada de almacenar los atributos de los distintos tipos de luces.

Tiene los siguientes **atributos**; **Ia** (color ambiental), **Id** (color difuso), **Is** (color especular), **position** (posición de la luz), **direction** (dirección de la luz), **angle** (ángulo del foco) y **exponent** (exponente de Warn), **at1** / **at2** / **at3** (atenuaciones de la luz), **applicator** (del tipo *PagLightAplication*, que será el encargado de aplicar el shader a los diferentes tipos de luz) y **type** (tipo de luz).

Sus **métodos** son:

Públicos: **getters and setters** de los distintos atributos.

2.9. PagLightAplication

Interfaz encargada de aplicar el shader a la luz. Esta será la clase padre de la que heredarán *PagAmbientLight*, *PagSpotLight* y *PagPuntualLight*.

No tiene atributos **atributos**.

Sus **métodos** son:

Públicos: **shaderAplicator**, que permite aplicar un shader que se pasará como primer parámetro (adsShader), será necesario pasarle la matriz de visión como segundo parámetro, ya que tanto posición como dirección tienen que pasarse a coordenadas de visión y **getType**, encargado de devolver un identificador para el tipo de luz.

2.10. PagAmbientLight

Clase encargada de la creación de luces ambientales.

Tiene los siguientes **atributos**; **light** (puntero a PagLightSource a través del cual se pasarán los atributos que corresponden a este tipo de luz: ia y la propia clase).

Sus **métodos** son:

Públicos: **shaderAplicator**, que permite aplicar un shader que se pasará como primer parámetro (adsShader), será necesario pasarle la matriz de visión como segundo parámetro, ya que tanto posición como dirección tienen que pasarse a coordenadas de visión y **getType**, encargado de devolver un identificador para el tipo de luz.

2.11. PagSpotLight

Clase encargada de la creación de luces tipo foco.

Tiene los siguientes **atributos**; **light** (puntero a PagLightSource a través del cual se pasarán los atributos que corresponden a este tipo de luz: id, is, position, at1, at2, at3 y la propia clase).

Sus **métodos** son:

Públicos: **shaderAplicator**, que permite aplicar un shader que se pasará como primer parámetro (adsShader), será necesario pasarle la matriz de visión como segundo parámetro, ya que tanto posición como dirección tienen que pasarse a coordenadas de visión y **getType**, encargado de devolver un identificador para el tipo de luz.

2.12. PagPuntualLight

Clase encargada de la creación de luces puntuales.

Tiene los siguientes atributos; **light** (puntero a PagLightSource a través del cuál se pasarán los atributos que corresponden a este tipo de luz: id, is, position, direction, angle, exponent, at1, at2, at3 y la propia clase).

Sus métodos son:

Públicos: **shaderAplicator**, que permite aplicar un shader que se pasará como primer parámetro (adsShader), será necesario pasarle la matriz de visión como segundo parámetro, ya que posición tiene que pasarse a coordenadas de visión y **getType**, encargado de devolver un identificador para el tipo de luz.

2.13. PagMaterial

Clase encargada de la creación de materiales para aplicar a un elemento. Se deduce que un elemento 3D siempre tendrá un material pero no siempre una textura, por este motivo, la textura formará parte del material como un puntero a PagTexture, inicialmente inicializado a 0.

Tiene los siguientes atributos; **texture** (puntero de la clase PagTexture, debido a que todo objeto tendrá un material asociado, sin embargo, este material podrá tener una textura o no tenerla y por defecto estará inicializada a 0), **Kd** (color difuso), **Ks** (color especular) y **shininess** (brillo).

Sus métodos son:

Públicos: **shaderApplicator**, que permite aplicar un shader que se pasará como parámetro (adsShader), **shaderApplicatorTexture** (activará las texturas) y métodos **getters** / **setters de los atributos**.

2.14. PagLibraryMaterial

Clase encargada de almacenar como una biblioteca los distintos materiales, asociados a un nombre.

Tiene los siguientes atributos; **instance** (instancia ya que se trata de un singleton) y **materials** (mapa de materiales almacenados por nombre).

Sus métodos son:

Públicos: **getInstance**, que devuelve la instancia del singleton, **insertMaterial**, que inserta un nuevo material, junto con su nombre, al mapa y **getMaterial**, que devuelve un material dado su nombre.

2.15. PagTexture

Clase encargada de la carga de texturas para aplicar a un elemento.

Tiene los siguientes atributos; **color** (unidad de textura del color que será enviada al sampler) y **shininess** (unidad de textura del brillo que será enviada al sampler).

Sus métodos son:

Privados: **invertImage**, que invierte la imagen de textura y **texParameters,** que aplicará los parámetros de textura. El cometido de estas funciones es no duplicar código en el constructor.

Públicos: **getColor**, que devuelve la unidad de textura color y **getBrillo,** que devuelve la unidad de textura color.

2.16. PagLibraryTexture

Clase encargada de almacenar como una biblioteca las distintas imágenes de textura, asociadas a un nombre.

Tiene los siguientes atributos; **instance** (instancia ya que se trata de un singleton) y **textures** (mapa de texturas almacenadas por nombre).

Sus métodos son:

Públicos: **getInstance**, que devuelve la instancia del singleton, **insertMaterial**, que inserta un nuevo material, junto con su nombre, al mapa y **getMaterial**, que devuelve un material dado su nombre.

2.17. PagRenderer

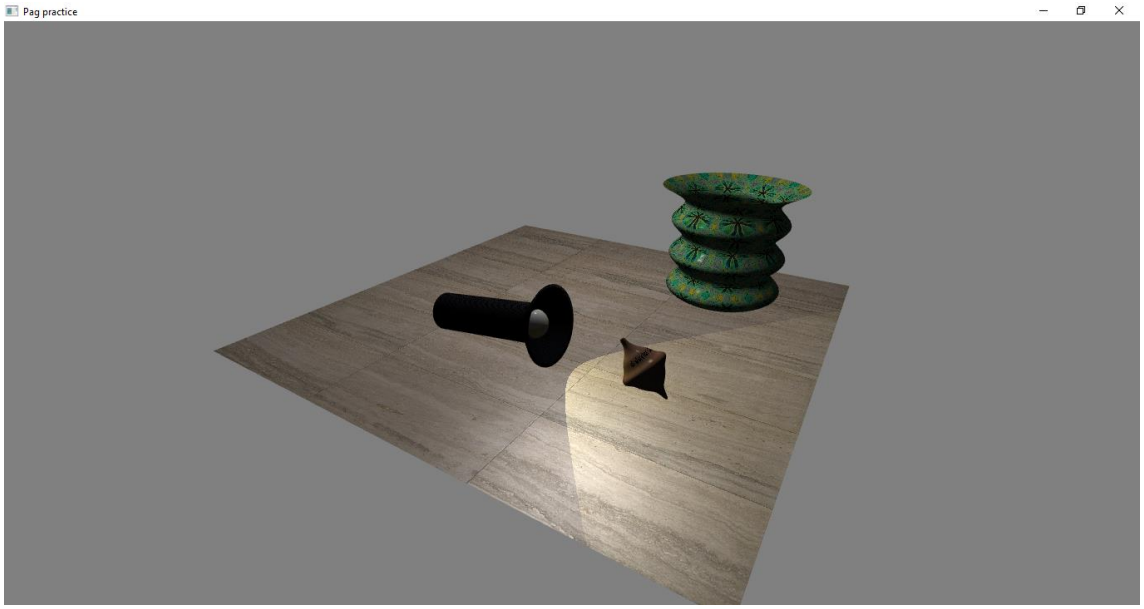
Clase encargada de renderizar.

Tiene los siguientes atributos; **instance** (única instancia Singleton de la clase *PagRenderer*), **escene** (escena que agrupará los distintos Pag3DElement), **camera** (cámara de la escena), **lights** (vector con las luces que iluminarán la escena), **material** (material que se aplicará al objeto de revolución), **pointShader** (shader de puntos), **trianglesShader** (shader de mallas de triángulos), **wiresShader** (shader de alambres), **debugShader** (shader del modo debug que incluye normales, tangentes y coordenadas de textura), **adsShader** (shader encargado de las luces ambiental, puntual y focal), **textureShader** (shader encargado de las luces ambiental, puntual y focal), **modeShader** (indica el shader que será usado), **modeDebug** (indica el modo del debugShader que se usa), **modeMov** (indica el modo en el que se moverá la cámara) y **modeLight** (indica el tipo de luz que se activará), **cameraChange** (cambia entre las dos posiciones que hay de cámara) y **cursorX** / **cursorY** (atributos para el control de posición del ratón para realizar movimientos).

Sus métodos son:

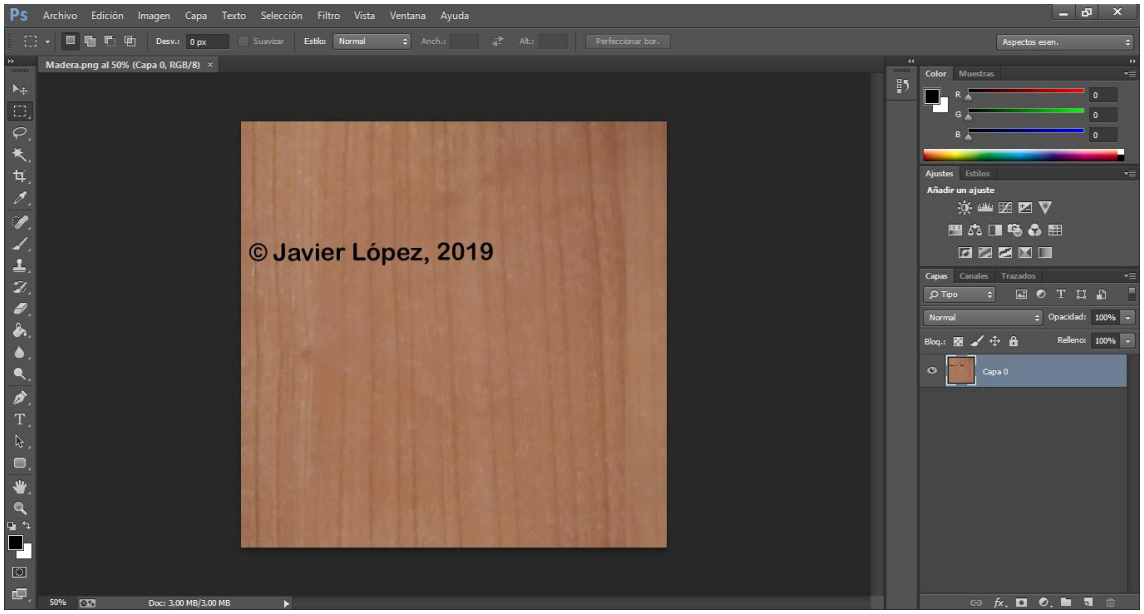
Públicos: **prepareOpenGL**, que prepara la escena inicializando, objetos, shaders, luces y materiales, **getInstance**, devuelve la única instancia *PagRenderer* y **callbacks**, métodos encargados del control de teclado, refrescar la ventana, movimientos del ratón…

3. CAPTURAS ESCENA FINAL Y DE PROCEDIMIENTO

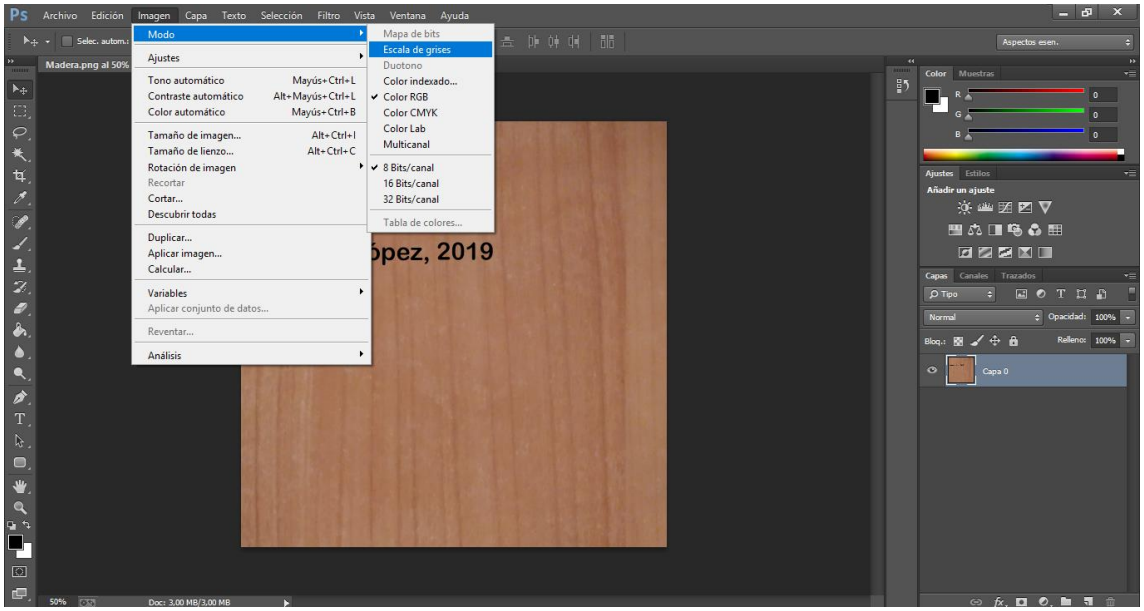


Proceso para generar una textura con Bump mapping:

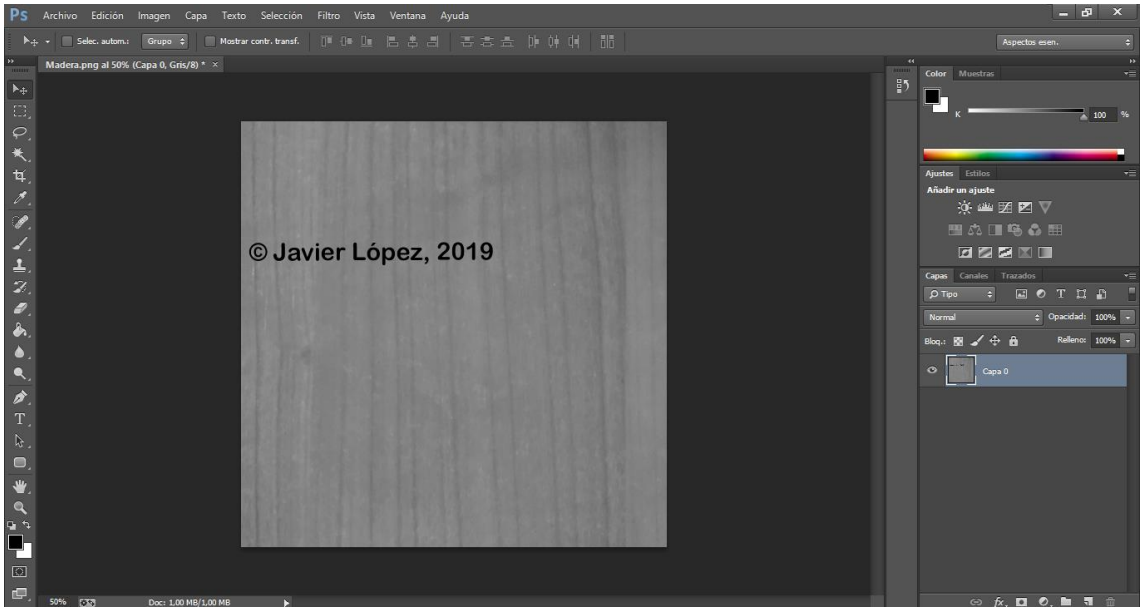
1.- Abrimos en Photoshop una textura madera (foto de mi escritorio) con el texto de copyright (que es donde se aplicará el bump mapping para que esta parte parezca más hundida, por eso está en negro). *Nota: para que sobresalga el texto debería estar en blanco*.



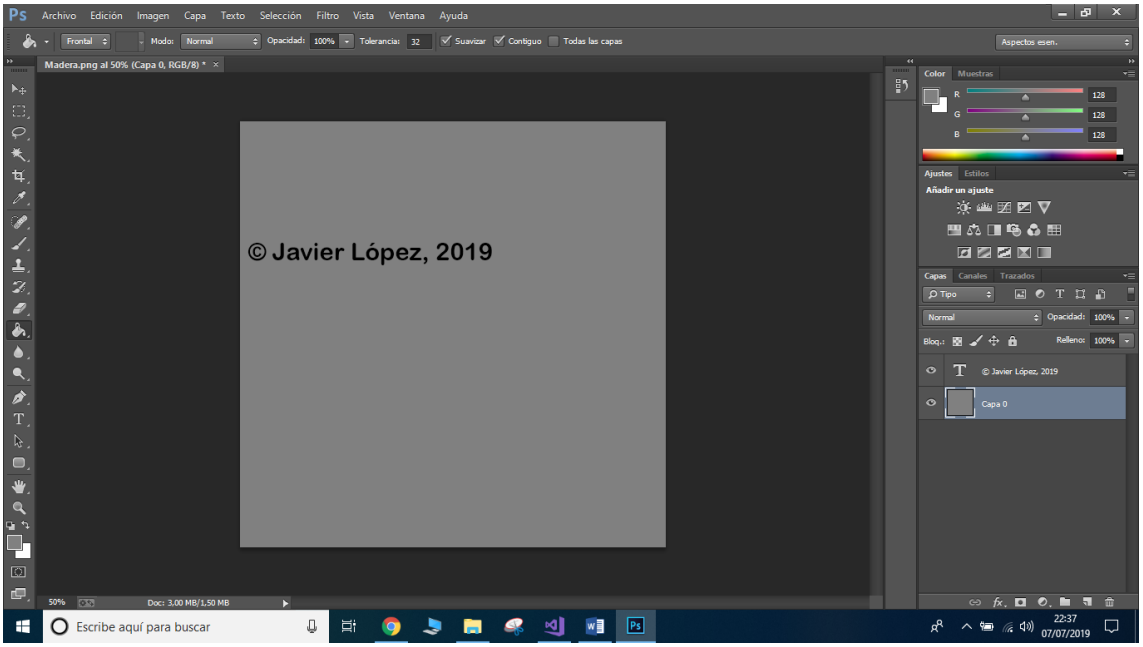
2.- Aplicamos una escala de grises



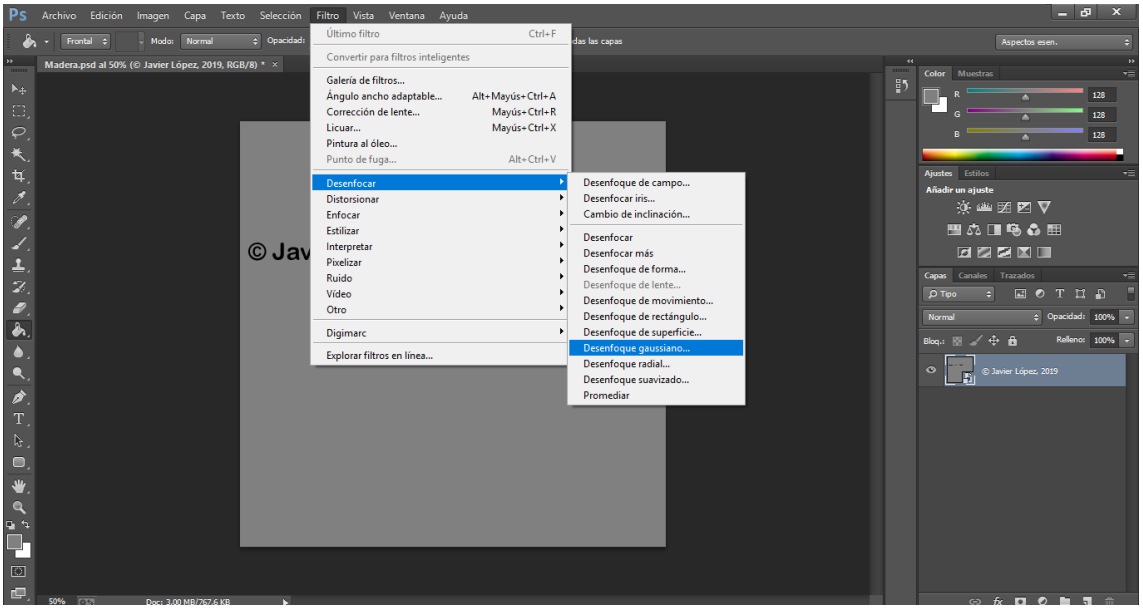
3.- Una vez tenemos la imagen en escala de grises la guardamos para el brillo. SE REPITE ESTE PROCESO CON TODAS LAS TEXTURAS, YA QUE SERÁN NECESARIAS PARA GENERAR EL BRILLO DE LAS TEXTURAS.

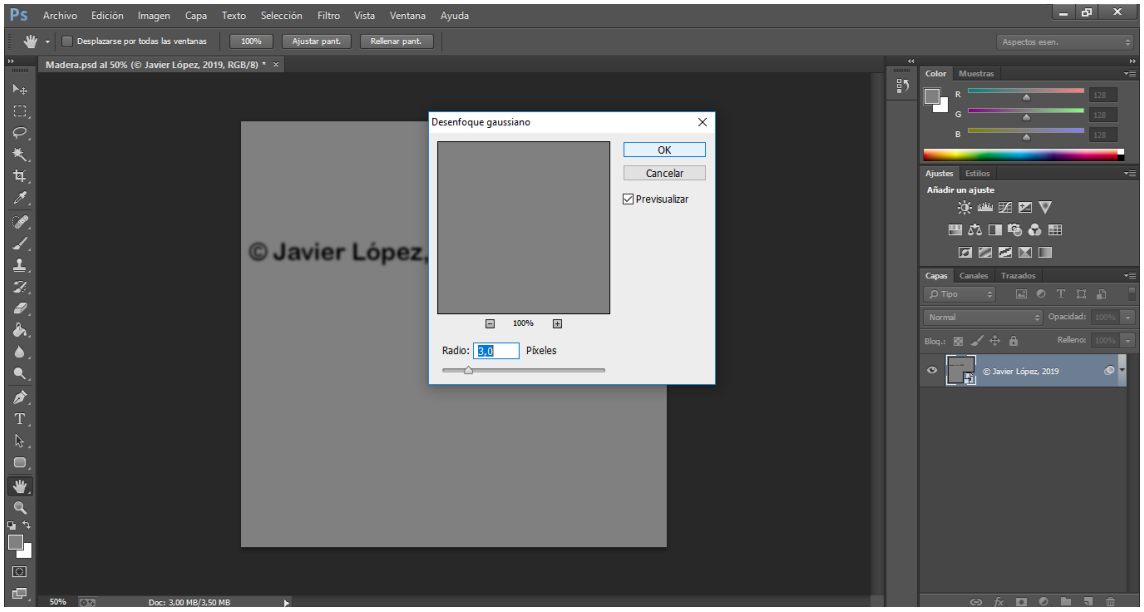


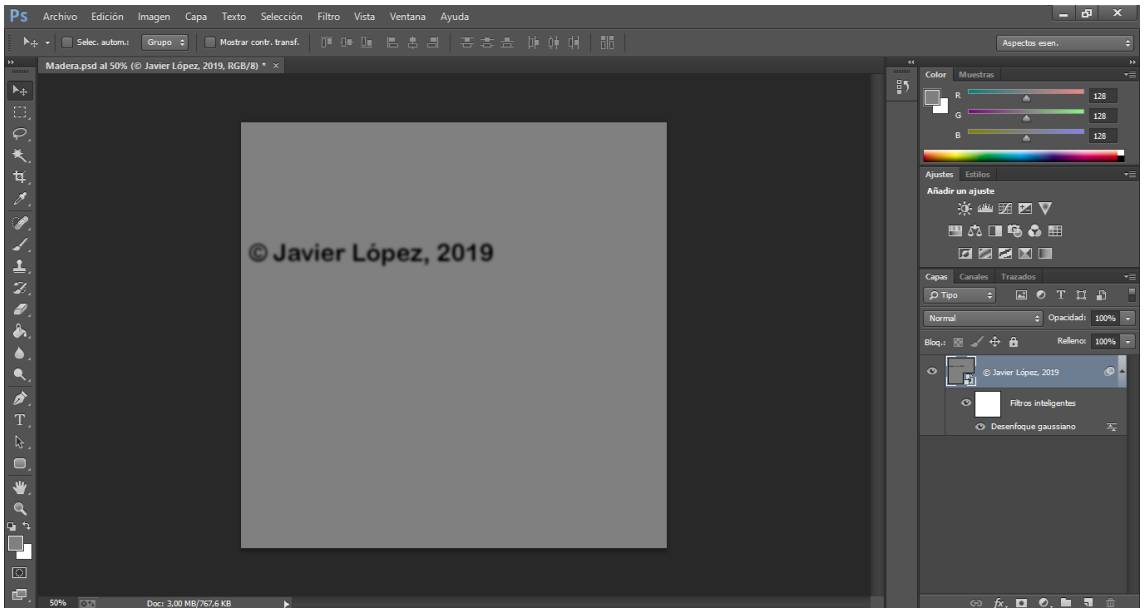
4.- Dejamos sólo la parte del copyright y le ponemos un fondo de gris neutro RGB 128



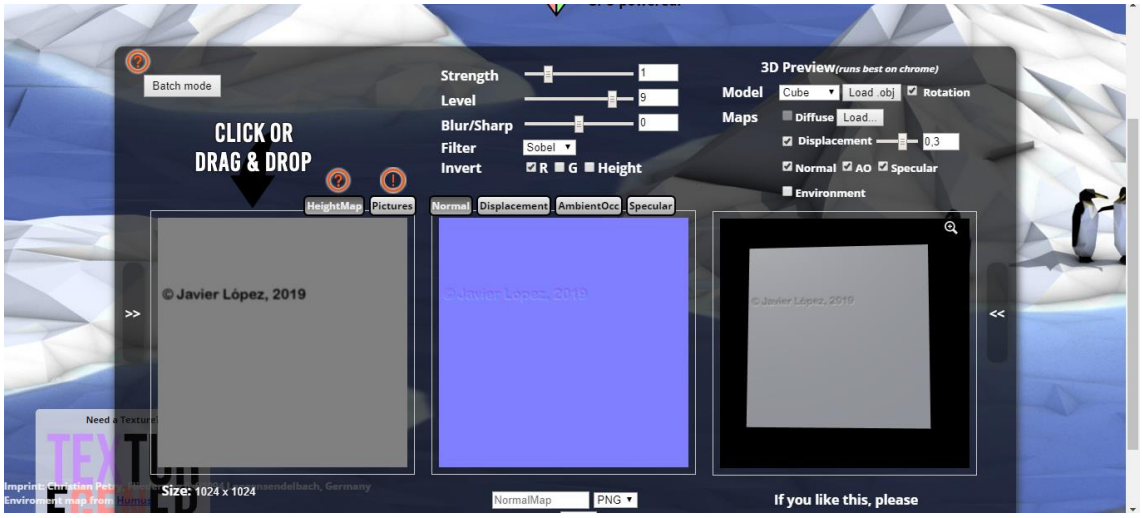
5.- Aplicamos un filtro Gaussiano de emborronamiento para que el relieve sea progresivo.

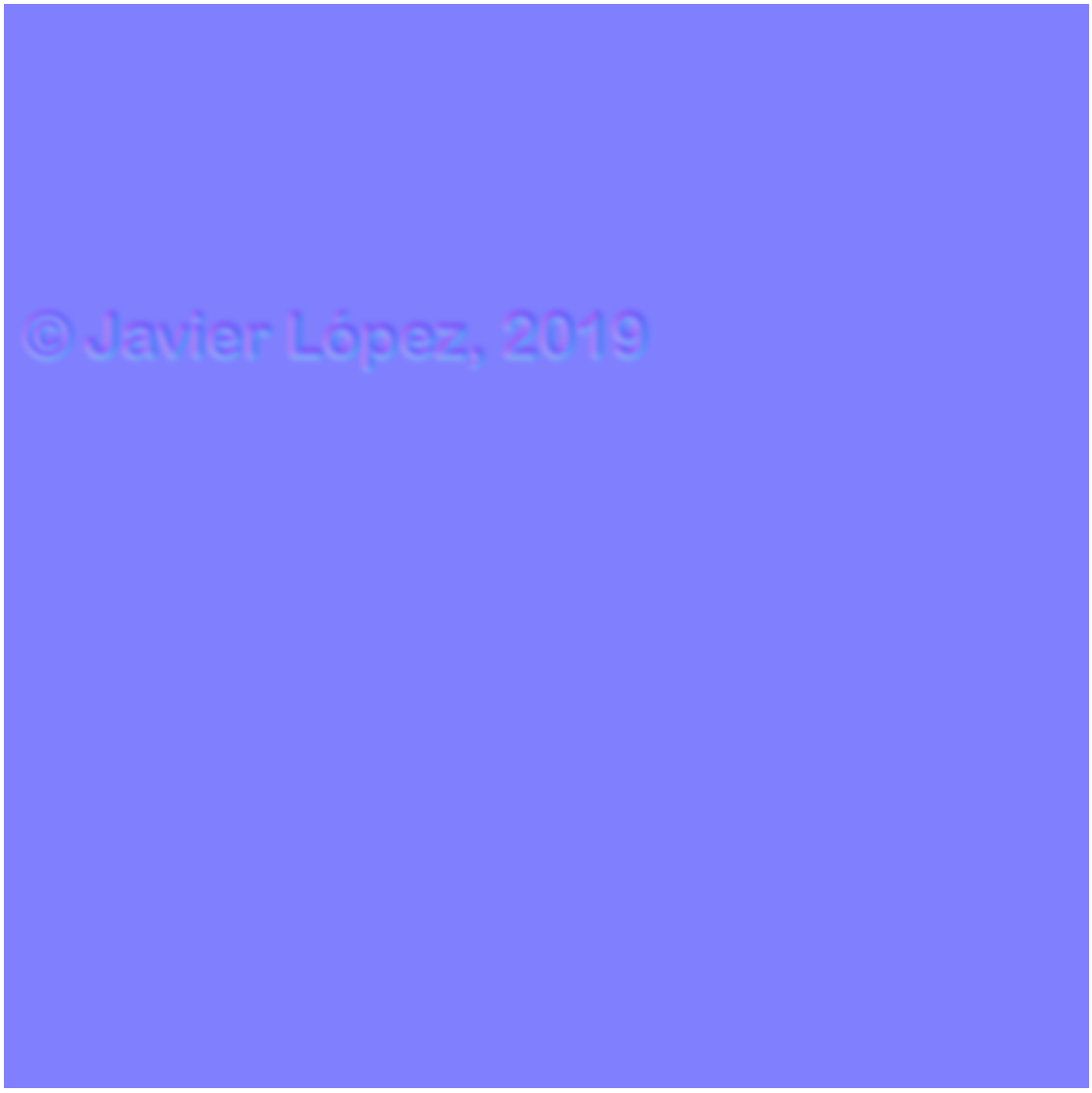






6.- Por último, una vez tenemos la imagen generamos su mapa de normales con la siguiente aplicación online (dándole los valores especificados en la página 21 del tema 8):





Textura final para bump mapping