

**Maxiwin**

**Documentación oficial.**

En este documento, a modo de manual para el programador, encontrará citada, de forma formal, todas las funcionalidades y usos comunes de esta librería. Las preguntas comunes serán contestadas por medio de los ejemplos o por medio de notas al pie de página, en caso de que su duda no pueda ser contestada por este medio, le invitamos a enviar su duda por medio del foro oficial de GitHub en el siguiente enlace: [https://...]

Recuerde que, como programador que lee esto, le invitamos a probar y ejecutar todos, o la mayoría, de los ejemplos que pueda llegar a encontrar a lo largo de este documento, antes de meterse de lleno con programación informal o a gran escala con el uso de esta librería.

Al ser Open Source y de código sin derechos de autor, se le garantizará que al programador final no reciba ninguna demanda por uso de material protegido por derechos de autor o similares, salvo que se intenté recrear el código copiándolo de forma descarada o intenté destruir la fuente del material original o medios de distribución oficial.

Toda la información necesaria de la librería la encontrará aquí.

* Silver Roberskull

**Índice**

|  |  |
| --- | --- |
| 2 | Introducción formal y Requisitos mínimos |
| 3 | Macros y funciones de automatización |
| 10 | Estructuras de datos |
| 21 | Funciones de llamada |
| 31 | Funciones de obtención de entradas |
| 37 | Funciones de control ventanal |
| 42 | Funciones de contexto gráfico |
| 45 | Funciones de dibujo primitivo |
| 52 | Funciones de dibujo avanzado |

**Introducción Formal**

**¿Qué es Maxiwin?**

Maxiwin se autodefine como una librería grafica orientada a C++, Open Source, Compacta, mínimamente dependiente y altamente portable.

**¿Cuál es el propósito de Maxiwin?**

Maxiwin está enfocada en ser una librería grafica ligera y de alto rendimiento, ideal para proyectos pequeños o amateur, aunque con algo de ingeniería a esta misma, y con el apoyo de implementaciones constantes por medio de nuevas actualizaciones, se pueden crear proyectos grandes con esta.

Su lectura en funciones y estructuras está enfocada en el idioma español, debido a una herencia de sintaxis de proyectos preliminares a este.

**¿Cómo surgió Maxiwin?**

Originalmente nació como una propuesta de desarrollo personal del usuario Silver Roberskull, luego frustrarse de que la librería Miniwin tuviera funciones muy limitadas en su momento, optando a seguir el siguiendo dilema: “Si no tengo las herramientas que yo quiero, entonces creare mis propias herramientas…”. Siguiendo un esquema de hiperoptimización y de pocas dependencias.

**Requisitos mínimos**

|  |  |
| --- | --- |
| Versión de C++ | 11 o superior |
| Librerías requeridas | * <windows.h> * <gdiplus.h> * <stdio.h> * <stdlib.h> |
| Sistema operativo destino | Windows 98, ME, 2000 o superior |
| Arquitectura destino | 32 y 64 bits |
| Nombre plantilla | ***maxiwin*** |
| Linker | ***-lwinmm -lgdi32 -lgdiplus*** |

**Macros Y Funciones de automatización**

|  |  |
| --- | --- |
| Nombre | Firma / Descripción |
| DynArray | |  | | --- | | template<typename T> class DynArray; | | Estructura de almacenamiento lineal de una lista ***T*** de elementos, su funcionamiento es equiparable al de ***std::vector<T>*** | |
| Callback | |  | | --- | | template<typename… Args> class Callback(Func F, Args&&… args); | | Estructura para almacenar, transportar y ejecutar un puntero a función (***Func F***) y sus argumentos (***Args&&… args***) fácilmente | |
| RGBA | |  | | --- | | Inline uint32\_t RGBA(uint32\_t r, uint32\_t g, uint32\_t b, uint32\_t a) | | Función pseudo macro que permite formar un color RGBA en formato ARGB fácilmente | |
| GetRed | |  | | --- | | inline uint32\_t GetRed(uint32\_t c) | | Obtiene el color rojo a partir de un color | |
| GetGreen | |  | | --- | | inline uint32\_t GetGreen(uint32\_t c) | | Obtiene el color verde a partir de un color | |
| GetBlue | |  | | --- | | inline uint32\_t GetBlue(uint32\_t c) | | Obtiene el color azul a partir de un color | |
| GetAlpha | |  | | --- | | inline uint32\_t GetAlpha(uint32\_t c) | | Obtiene la transparencia a partir de un color | |
| rgba | Macro sinónimo de ***RGBA*** |
| rgb | Macro sinónimo de ***RGBA(r,g,b,255)*** |
| RGB | Macro preventiva de definición, sinónimo de ***RGBA(r,g,b,255)*** |
| mezclar\_colores | |  | | --- | | inline uint32\_t mezclar\_colores(uint32\_t A, uint32\_t B) | | Función para mezclar dos colores, a partir de su nivel de transparencia | |
| multiplicar\_colores | |  | | --- | | inline uint32\_t multiplicar\_colores(uint32\_t A, uint32\_t B) | | Función para multiplicar dos colores | |
| colores | |  | | --- | | namespace colores | | Namespace que contiene una extensa lista de colores, listos para usarse. Estos son los colores de susodicha lista:   |  | | --- | | * MascaraAlpha * AzulAlicia * BlancoAntiguo * Agua * Aguamarina * AzulCeleste * Beige * Bizcocho * Negro * AlmendraBlanqueada * Azul * AzulVioleta * Marron * MaderaRustica * AzulCadete * Chartreuse * Chocolate * Coral * AzulAciano * SedaDeMaIz * CarmesI * Cian * AzulOscuro * CianOscuro * OroViejoOscuro * GrisOscuro * VerdeOscuro * CaquiOscuro * MagentaOscuro * VerdeOlivaOscuro * NaranjaOscuro * OrquIdeaOscura * RojoOscuro * SalmonOscuro * VerdeMarOscuro * AzulPizarraOscuro * GrisPizarraOscuro * TurquesaOscuro * VioletaOscuro * RosaIntenso * AzulCieloIntenso * GrisTenue * AzulDodger * LadrilloRefractario * BlancoFloral * VerdeBosque * Fucsia * GrisGainsboro * BlancoFantasma * Oro * VaraDeOro * Gris * Verde * VerdeAmarillo * RocIoDeMiel * RosaCaliente * RojoIndio * Indigo * Marfil * Caqui * Lavanda * RuborLavanda * VerdeCesped * ChiffonDeLimon * AzulClaro * CoralClaro * CianClaro * AmarilloOroClaro * GrisClaro * VerdeClaro * RosaClaro * SalmonClaro * VerdeMarClaro * AzulCieloClaro * GrisPizarraClaro * AzulAceroClaro * AmarilloClaro * Lima * VerdeLima * Lino * Magenta * Granate * AguamarinaMedia * AzulMedio * OrquIdeaMedia * PurpuraMedia * VerdeMarMedio * AzulPizarraMedio * VerdePrimaveraMedio * TurquesaMedia * RojoVioletaMedio * AzulMedianoche * CremaDeMenta * RosaBrumoso * MocasIn * BlancoNavajo * AzulMarino * EncajeAntiguo * Oliva * VerdeOliva * Naranja * RojoNaranja * OrquIdea * OroPalido * VerdePalido * TurquesaPalido * RojoVioletaPalido * PapayaSuave * MelocotonSuave * Peru * Rosa * Ciruela * AzulTalco * Purpura * Rojo * MarronRosado * AzulReal * MarronMontura * Salmon * MarronArenoso * VerdeMar * ConchaDeMar * Siena * Plata * AzulCielo * AzulPizarra * GrisPizarra * Nieve * VerdePrimavera * AzulAcero * Bronceado * VerdeAzulado * Cardo * Tomate * Transparente * Turquesa * Violeta * Trigo * Blanco * HumoBlanco * Amarillo * AmarilloVerde | | |
| mensajeria | |  | | --- | | namespace mensajeria | | Namespace que contiene una lista de modos para los mensajes, Estos son los modos de susodicha lista:   * BOTN\_OK * BOTN\_OK\_CANCELAR * BOTN\_ABORTAR\_REINTENTAR\_IGNORAR * BOTN\_SI\_NO\_CANCELAR * BOTN\_SI\_NO * BOTN\_REINTENTAR\_CANCELAR * BOTN\_CANCELAR\_INTENTAR\_CONTINUAR * ICON\_CRUZ * ICON\_PREGUNTA * ICON\_EXCLAMACION * ICON\_ASTERISCO * ICON\_USUARIO * ICON\_ADVERTENCIA * ICON\_ERROR * ICON\_INFORMACION * ICON\_PARAR * DEFT\_BOTON\_1 * DEFT\_BOTON\_2 * DEFT\_BOTON\_3 * DEFT\_BOTON\_4 * MODL\_APP * MODL\_SISTEMA * MODL\_TAREA * MODL\_AYUDA * MODL\_NOFOCUS * CTRL\_FOREGROUND * CTRL\_DEFAULT\_ESCRITORIO * CTRL\_MASALTO * CTRL\_DERECHA * CTRL\_LECTURA\_DERECHA * CTRL\_NOTIFICACION * CTRL\_NOTIFICACION\_NT3X * MASK\_BOTN * MASK\_ICON * MASK\_DEFT * MASK\_MODL * MASK\_CTRL | |
| botoneria | |  | | --- | | namespace botoneria | | Namespace que se encuentra dentro del namespace ***mensajeria***, que posee una breve lista de traducción de entradas del usuario por medio de los mensajes:   * CRUZ * OK * CANCELAR * ABORTAR * REINTENTAR * IGNORAR * SI * NO * INTENTAR * CONTINUAR | |
| teclas | |  | | --- | | enum teclas | | Enumerador que enumera teclas de función y acción, que posee una breve lista de dichas teclas:   * NINGUNA * ESCAPE * IZQUIERDA * DERECHA * ARRIBA * ABAJO * F1 * F2 * F3 * F4 * F5 * F6 * F7 * F8 * F9 * F10 * F11 * F12 * ESPACIO * RETURN * BACKSPACE * SHIFT * CTRL * ALT * TAB * MAYUSCULAS * SUPR * LWIN * RWIN * APP | |
| fuente\_estilos | |  | | --- | | enum class fuente\_estilos : unsigned int | | Enumerador que enumera las posibles combinaciones de estilos, las cuales puedes apilar con el operador binario ***or*** (‘|’):   * NEGRITA * SUBRAYADA * ITALICA | |

**Estructuras de datos**

**Img**

**Proposito**

El propósito de esta estructura es almacenar de forma sencilla la información en bruto de una imagen, en un formato que recordaría al del formato de archivo ***.bmp***.

**Variables publicas**

|  |  |
| --- | --- |
| unsigned int ancho | Ancho de la imagen |
| unsigned int alto | Alto de la imagen |
| uint32\_t\* contenido | Lista de pixeles, su longitud es determinada por ***ancho \* alto*** |

**Funciones y métodos**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Constructor | Existen diferentes formas de usarlo, las existentes son:   |  | | --- | | img() | | img(unsigned int w, unsigned int h) | | img(unsigned int w, unsigned int h, uint32\_t\* c) | | img(const img& other) | | img(img&& other) noexcept | |
| Destructor | Sí, posee uno |
| Operadores | Posee:   |  | | --- | | * Igualación * Igualador por movimiento * Comparación igualitaria * Comparación desigualitaria * ***valido()***: Devuelve ***true***, si la imagen posee contenido. * ***contiene(color)***: Devuelve ***true***, si la imagen posee dicho ***color***. * ***contiene\_unicamente(color)***: Devuelve ***true***, si la imagen posee únicamente dicho ***color***. Devolverá ***false***, si posee otro color que no sea ***color*** o si directamente no lo posee. | |
| ***at(int x, int y)*** | Retorna la referencia al pixel especificado, solo funcionará si el pixel a consultar esta dentro del rango de la imagen |
|  |  |
| ***remplazar(uint32\_t \_dst, uint32\_t \_src)*** | Remplazará el color ***\_dst*** con el color ***\_src***. |
| ***espejo(bool horz, bool vert)*** | Voltea en espejo la imagen: ***horz*** para un volteo horizontal ***vert*** para un volteo vertical |
| ***recorte(int x, int y, int w, int h)*** | Retorna una nueva imagen con el área recortada de la imagen origen. |
| ***estirar(int w, int h)*** | Cambia el tamaño del ***ancho*** y ***alto***, estirando el ***contenido*** de la imagen en el proceso. |
| ***sesgar(float skewX, float skewY)*** | Cambia el tamaño de ***ancho***, ***alto*** y adapta el ***contenido*** de la imagen en función de un sesgado. |
| ***rotar(float angleDegrees)*** | Cambia el tamaño de ***ancho***, ***alto*** y adapta el ***contenido*** de la imagen en función de una rotación. |

**Anim**

**Proposito**

El propósito de esta estructura es de almacenar una imagen, dividida por fotogramas, cuyos mismos se almacenan en el formato antes mencionado para ***img***.

**Variables publicas**

|  |  |
| --- | --- |
| unsigned int ancho | Ancho de la animación. |
| unsigned int alto | Alto de la animación. |
| unsigned int frame | Frame actual al que está consultando la animación. |
| unsigned int cantidad | Cantidad de fotogramas dentro de la animación. |
| uint32\_t\*\* contenido | Lista de lista de pixeles, la lista principal almacena los fotogramas, las sublistas almacenan los pixeles de esos fotogramas. Su longitud es determinada por ***cantidad***, mientras que la profundidad está determinada por ***ancho \* alto***. |

**Funciones y métodos**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Constructor | Existen diferentes formas de usarlo, las existentes son:   |  | | --- | | anim() | | anim(unsigned int w, unsigned int h) | | anim(unsigned int w, unsigned int h, unsigned int c) | | anim(const anim& other) | | anim(anim&& other) noexcept | | anim(const img& \_ig, unsigned int count = 0U) | | anim(unsigned int w, unsigned int h, uint32\_t\* cc, unsigned int c) | | anim(unsigned int w, unsigned int h, uint32\_t\*\* cc, unsigned int c) | |
| Destructor | Sí, posee uno. |
| Operadores | Posee:   |  | | --- | | * Selector arreglado (***[x]***): Retorna una nueva imagen, producto de un fotograma elegido. * Igualación. * Igualación por movimiento. * Suma relativa: Para ir acumulando fotogramas. * Suma. * Comparación igualitaria. * Comparación desigualitaria. * ***valido()***: Devuelve ***true***, si la animación posee fotogramas, y susodichos fotogramas poseen contenido. * ***agregar\_frame(const img&)***: Añade el contenido de la imagen como si fuera un nuevo fotograma. El nuevo fotograma irá al final de la cola. * ***agregar\_frame(const img&, const unsigned int& c)***: Añade nuevos fotogramas, nacidos a partir de copias de la imagen. Los nuevos fotogramas irán al final de la cola. * ***agregar\_frame(const anim&)***: Añade los fotogranas de la imagen origen a la imagen destino. Los nuevos fotogramas irán al final de la cola. * ***eliminar\_frame(size\_t x, size\_t n)***: Elimina la cantidad de fotogramas indicados en ***n***, a partir de la posición ***x***. Se eliminará un solo fotograma por defecto, almenos que indiques manualmente el valor de ***n***. * ***limpiar\_frames()***: Borra todos los fotogramas almacenados en la animación, sin excepciones. Cabe aclarar que esto invalidará la animación, ya que una animación vacía es sinónimo de invalida. | |
| ***at(size\_t n, int x, int y)*** | Retornará la referencia al pixel (***x*** e ***y***) del fotograma ***n***. Solo funcionará si el pixel seleccionado está dentro del área del fotograma indicado, y el fotograma a consultar no es uno que se encuentra fuera de la cola de fotogramas de la animación. |
| ***segmento(unsigned int desde, unsigned int \_con)*** | Retorna una nueva animación, producto del segmento desde ***desde***, hasta ***desde + \_con***. |
| ***siguiente\_frame(unsigned int \_n)*** | Retornará la referencia a esta misma animación, lo único que hará será que la animación recorra una cantidad ***\_n*** de fotogramas. Por defecto, solo será uno solo. |
| ***anterior\_frame(unsigned int \_n)*** | Retornará la referencia a esta misma animación, lo único que hará será que la animación recorra inversamente una cantidad ***\_n*** de fotogramas. Por defecto, solo será uno solo. |
| ***estirar(int w, int h)*** | Retornará la referencia a esta misma animación, lo único que hará será cambiar el tamaño de la animación, estirando el contenido de todos los fotogramas. |

**Fnt**

**Propósito**

El propósito de esta estructura es almacenar una referencia a una fuente de texto, si bien no almacena la fuente de texto como tal, sí que almacena el formato configurado para la fuente, además de su nombre.

**Variables publicas**

|  |  |
| --- | --- |
| DynArray<char> nombre | Nombre de la fuente de texto |
| int dimension | Tamaño de la fuente |
| int angulo | Angulo de inclinación de la fuente |
| bool italica | De ser ***true***, la fuente poseerá cursiva |
| bool subrayada | De ser ***true***, la fuente poseerá subrayado |
| bool negrita | De ser ***true***, la fuente poseerá negrita |
| bool monoespacio | De ser ***true***, la fuente poseerá monoespaciado |

**Funciones y métodos**

|  |  |  |  |
| --- | --- | --- | --- |
| Constructor | Existen diferentes formas de usarlo, las existentes son:   |  | | --- | | fnt() | | fnt(const DynArray<char>& \_name) | |

**Pnt**

**Propósito**

Almacenar un punto en el espacio, utilizando 3 unidades dimensionales (3D).

**Variables publicas**

|  |  |
| --- | --- |
| float x | Eje en X |
| float y | Eje en Y |
| float z | Eje en Z |

**Funciones y métodos**

|  |  |
| --- | --- |
| Constructor | Existen diferentes formas de usarlo, las existentes son:  pnt(float \_x = 0, float \_y = 0, float \_z = 0) |
| Operadores | Posee:   |  | | --- | | * Comparación igualitaria * Comparación desigualitaria * Inversión de signo * Suma relativa * Suma * Resta relativa * Resta * Multiplicación relativa * Multiplicación * División relativa * División | |
| ***static dot(const pnt& a, const pnt& b)*** | Calcula el producto escalar entre ***a*** y ***b***. |
| ***dot(const pnt& o)*** | Calcula el producto escalar entre sí mismo y ***o***. |
| ***static cruz(const pnt& a, const pnt& b)*** | Calcula el producto vectorial entre ***a*** y ***b***. |
| ***cruz(const pnt& o)*** | Calcula el producto vectorial entre sí mismo y ***o***. |
| ***distancia(const pnt& o)*** | Calcula la distancia trigonométrica entre sí mismo y ***o***. |
| ***longitud()*** | Calcula la longitud del vector al origen |
| ***normalizar()*** | Calcula la dirección de la normal con respecto al origen |

**Mdl**

**Propósito**

Poder almacenar un modelo tridimensional, inclusive sus texturas, pivote y si le afecta el brillo y/o posee iluminación suave o no. Es, en comparación a las estructuras anteriores, la más compleja hasta el momento.

**Variables publicas**

|  |  |
| --- | --- |
| anim textura | Texturas del modelo 3D, cada fotograma es una textura individual para el modelo |
| pnt pivote | Pivote del centro del modelo 3D, influye en su renderizado |
| bool brillo = ***true*** | “¿Le afectará el brillo?”.  Valor por defecto: ***true*** |
| bool suavizado = ***false*** | “¿La influencia del brillo será suavizada?”.  Valor por defecto: ***false*** |

**Funciones y métodos**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Constructor | Existen diferentes formas de usarlo, las existentes son:   |  | | --- | | mdl() | | mdl(const mdl& o) | | mdl(mdl&& o) noexcept | | mdl(int mt, float escala = 1.0f, anim aplicar\_textura = {}) | | mdl(PRIMITIVA mt, float escala = 1.0f, anim aplicar\_textura = {}) | |
| Destructor | Si, posee uno. |
| Operadores | Posee:   |  |  |  | | --- | --- | --- | | * Igualación * Igualación por movimiento * Comparación igualitaria * Comparación desigualitaria * Suma relativa:  |  | | --- | | * Con otro ***mdl:*** Para añadir vértices, caras y UVs. * Con un ***pnt***: Para mover los vértices. |  * Suma:  |  | | --- | | * Con otro ***mdl:*** Para añadir vértices, caras y UVs. * Con un ***pnt***: Para mover los vértices. |  * Resta relativa: Con un ***pnt***, para mover los vértices. * Resta: Con un ***pnt***, para mover los vértices. * Multiplicación relativa: Con un ***float***, para escalar los vértices. * Multiplicación: Con un ***float***, para escalar los vértices. * División relativa: Con un ***float***, para escalar los vértices. * División: Con un ***float***, para escalar los vértices. * ***textura\_valida()***: Devuelve ***true***, si todas las texturas son válidas. * ***textura\_valida(size\_t x)***: Devuelve ***true***, si la textura seleccionada es válida. * ***datos\_vertices()***: Devuelve el puntero correspondiente al arreglo de vértices. * ***datos\_longitudes\_faces()***: Devuelve el puntero correspondiente al arreglo de longitudes de cada una de las caras. * ***datos\_faces()***: Devuelve el puntero correspondiente al arreglo en bruto de las caras apiladas secuencialmente. * ***datos\_uvs()***: Devuelve el puntero correspondiente al arreglo en bruto de las UVs apiladas secuencialmente, cada UV requiere decodificación previa antes de usar. * ***agregar\_vertice(const pnt& v)***: Agrega un vértice. * ***agregar\_vertice(const float& \_x, const float& \_y, const float& \_z)***: Agrega un vértice. * ***cantidad\_vertices()***: Devuelve la cantidad de vértices del modelo. * ***obtener\_vertice(size\_t \_x):***  Devuelve el vértice seleccionado. * ***modificar\_vertice(size\_t \_x, const pnt& \_new\_dot)***: Selecciona un vértice, para cambiar su valor. * ***eliminar\_vertice(size\_t \_x, size\_t \_n)***: Elimina una cantidad de vértices ***\_n*** a partir de ***\_x***, la cantidad a eliminar por defecto es uno. * ***limpiar\_vertices()***: Elimina todos los vértices, sin excepciones. * ***agregar\_cara(size\_t i1, size\_t i2, pnt tp1, pnt tp2)***: Agrega una cara de dos lados, junto con su UV. * ***agregar\_cara(size\_t i1, size\_t i2, size\_t i3, pnt tp1, pnt tp2, pnt tp3)***: Agrega una cara de tres lados, junto con su UV. * ***agregar\_cara(size\_t i1, size\_t i2, size\_t i3, size\_t i4, pnt tp1, pnt tp2, pnt tp3, pnt tp4)***: Agrega una cara de cuatro lados, junto con su UV. * ***cantidad\_caras()***: Obtiene la cantidad de caras. * ***longitud\_cara(size\_t \_x)***: Obtiene la longitud de una cara ***\_x***, siendo esta la cantidad de lados que esta posea. * ***obtener\_cara(size\_t \_face, int \_dot)***: Obtiene la dirección de vértice al que apunta la esquina ***\_dot*** de la cara ***\_face***. * ***obtener\_uv(size\_t \_face, int \_uv)***: Obtiene la UV de la esquina ***\_uv*** de la cara ***\_face***. * ***modificar\_cara(size\_t \_face, size\_t i1, size\_t i2)***: Cambia la dirección de vértice de cada esquina de la cara ***\_face***, inclusive cambia la cantidad de esquinas que posea esta cara a dos. * ***modificar\_cara(size\_t \_face, size\_t i1, size\_t i2, size\_t i3)***: Cambia la dirección de vértice de cada esquina de la cara ***\_face***, inclusive cambia la cantidad de esquinas que posea esta cara a tres. * ***modificar\_cara(size\_t \_face, size\_t i1, size\_t i2, size\_t i3, size\_t i4)***: Cambia la dirección de vértice de cada esquina de la cara ***\_face***, inclusive cambia la cantidad de esquinas que posea esta cara a cuatro. * ***modificar\_uv(size\_t \_face, pnt \_new\_tp1, pnt \_new\_tp2)***: Cambia las UVs de cada esquina de la cara ***\_face***, inclusive cambia la cantidad de esquinas que posea esta cara a dos. * ***modificar\_uv(size\_t \_face, pnt \_new\_tp1, pnt \_new\_tp2, pnt \_new\_tp3)***: Cambia las UVs de cada esquina de la cara ***\_face***, inclusive cambia la cantidad de esquinas que posea esta cara a tres. * ***modificar\_uv(size\_t \_face, pnt \_new\_tp1, pnt \_new\_tp2, pnt \_new\_tp3, pnt \_new\_tp4)***: Cambia las UVs de cada esquina de la cara ***\_face***, inclusive cambia la cantidad de esquinas que posea esta cara a cuatro. * ***eliminar\_cara(size\_t \_x, size\_t \_n)***: Elimina una cantidad de caras ***\_n*** a partir de ***\_x***, la cantidad de caras a eliminar por defecto es de una. * ***limpiar\_caras()***: Elimina todas las caras del modelo, sin excepciones. | |
| ***normal\_de(size\_t \_x)*** | Devuelve la normal a que apunta una cara ***\_x***. |
| ***rellenar\_textura()*** | Cambia la posición de todas las UVs para que sus respectivas caras puedan cubrir toda la textura, sin excepciones. |
| ***trasladar(const pnt& delta)*** | Mueve todos los vértices relativamente a la posición ***delta***. Retorna la referencia al modelo. |
| ***escalar(float factor)*** | Escala todos los vértices relativamente según el escalar ***factor***. Retorna la referencia al modelo. |
| ***rotarX(float rad)*** | Rota todos los vértices relativamente en el eje X según el escalar ***rad***. Retorna la referencia al modelo. |
| ***rotarY(float rad)*** | Rota todos los vértices relativamente en el eje Y según el escalar ***rad***. Retorna la referencia al modelo. |
| ***rotarZ(float rad)*** | Rota todos los vértices relativamente en el eje Z según el escalar ***rad***. Retorna la referencia al modelo. |

**Cam**

**Propósito**

Almacenar una serie de configuraciones para la perspectiva de renderizado de modelos 3D.

**Variables globales**

|  |  |
| --- | --- |
| pnt posicion = ***{0,0,0}*** | Posicion de la cámara  Valor por defecto: ***{0,0,0}*** |
| pnt objetivo = ***{0,0,1}*** | Dirección a la que mira la cámara  Valor por defecto: ***{0,0,1}*** |
| pnt up = ***{0,1,0}*** | Dirección del cielo para la cámara  Valor por defecto: ***{0,1,0}*** |
| float fov = ***90.0f*** | Distorsión de enfoque  Valor por defecto: ***90.0f*** |
| float z\_near = ***0.0f*** | Plano cercano, distancia mínima  Valor por defecto: ***0.0f*** |
| float z\_far = ***1000.0f*** | Plano lejano, distancia máxima  Valor por defecto: ***1000.0f*** |

**Funciones y métodos**

|  |  |  |  |
| --- | --- | --- | --- |
| Constructores | Existen diferentes formas de usarlo, las existentes son:   |  | | --- | | cam() | | cam(pnt pos, pnt obj = {0, 0, 1}, float \_fov = 90.0f) | |
| ***focal()*** | Devuelve la distancia de punto focal, traduciendo ***objetivo*** a un valor lineal. |

**Funciones de llamada**

**Mensaje**Firma:

void mensaje(const DynArray<char> msj, const DynArray<char> title = "Mensaje...", int method = 0)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***msj*** | const DynArray<char> | Mensaje a mostrar |  |
| ***title*** | const DynArray<char> | Título del mensaje | ***“Mensaje…”*** |
| ***method*** | int | Configuraciones del mensaje | ***0*** |

**Descripción**

Abre una ventana emergente con un mensaje (***msj***) y un título (***title***), su diseño variará dependiendo de cómo lo configuraste (***method***).

Mientras la ventana emergente este abierta, el programa esperara a que sea cerrada para seguir ejecutando código.

**Valor de retorno**

void

**Ejemplo de uso**

|  |
| --- |
| #include"maxiwin.h"  usign namespace maxiwin;  int main(){  // Mensaje convencional  mensaje("Hola a todos!");  // Mensaje a partir de un texto variable y con un titulo  char menmen[15];  sprintf(menmen, "Mi DNI es: ");  sprintf(menmen + 11, "%d", 1234);  mensaje(menmen, "DNI");  // Mensaje con formato personalizado  mensaje("Ese no era, perdon.", "ERROR!", 0x10);  return 0;  } |
| Primer mensaje:    Segundo mensaje:    Tercer mensaje:    ... El programa finaliza. |

**Pregunta**

Firma:

bool pregunta(const DynArray<char> msj, const DynArray<char> title = "Pregunta...", int method = 0, int result = 0)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***msj*** | const DynArray<char> | Mensaje a mostrar |  |
| ***title*** | const DynArray<char> | Título del mensaje | ***“Pregunta…”*** |
| ***method*** | int | Configuración del mensaje | ***0*** |
| ***result*** | int | Valor esperado | ***0*** |

**Descripción**

Abre una ventana emergente con un mensaje (***msj***) y un título (***title***), su diseño variará dependiendo de cómo lo configuraste (***method***).

Mientras la ventana emergente este abierta, el programa esperara a que sea cerrada para seguir ejecutando código.

El valor a retornar es proporcional a si la respuesta del usuario es igual a la indicada por la función de origen (***result***).

**Valor de retorno**

bool

|  |  |  |
| --- | --- | --- |
| Valor | Condición | Caso |
| ***true*** | … Si … | La respuesta es igual a ***result*** |
| ***false*** | … Si … | La respuesta no es igual a ***result*** |

**Ejemplo de uso**

|  |
| --- |
| #include"maxiwin.h"  usign namespace maxiwin;  int main(){  // Una pregunta simple  bool awser = pregunta("La torre Eiffel mide 10 metros?", "Te hago una consulta...", mensajeria::BOTN\_SI\_NO, mensajeria::botoneria::SI);  if(awser) mensaje("INCORRECTO!", "ERROR!", mensajeria::ICON\_ERROR);  else mensaje("Muy bien!", "Sigue asi maquina!");  return 0;  } |
| La pregunta:    ... En caso de contestar “Sí”:    ... En caso de contestar “No”:    ... El programa finaliza. |

**Consulta**

Firma:

int consulta(const DynArray<char> msj, const DynArray<char> title = "Consulta...", int method = 0)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***msj*** | const DynArray<char> | Mensaje a mostrar |  |
| ***title*** | const DynArray<char> | Título del mensaje | ***“Consulta…”*** |
| ***method*** | int | Configuración del mensaje | ***0*** |

**Descripción**

Abre una ventana emergente con un mensaje (***msj***) y un título (***title***), su diseño variará dependiendo de cómo lo configuraste (***method***).

Mientras la ventana emergente este abierta, el programa esperara a que sea cerrada para seguir ejecutando código.

El valor a retornar es proporcional la respuesta del usuario.

**Valor de retorno**

Int

Proporcional a la respuesta del usuario, se recomienda consultar con ***maxiwin::mensajeria::botoneria***.

**Ejemplo de uso**

|  |
| --- |
| #include"maxiwin.h"  usign namespace maxiwin;  int main(){  // Preguntar en bucle  int foo = 0;  while(foo != mensajeria::botoneria::SI) foo = consulta("Diga \"Si\".", "Consulta...", mensajeria::BOTN\_SI\_NO);  return 0;  } |
| Hasta que no contestemos que “Sí”, dira:    ... Una vez que digamos “Sí”, finalizará el programa. |

**Espera**

Firma:

void espera(int miliseg)

**Parametros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***miliseg*** | int | Cantidad de milisegundos a esperar |  |

**Descripción**

Le indica al programa que espere una ***miliseg*** cantidad de milisegundos. Durante este periodo, el programa no ejecutará nada de código siguiente hasta el tiempo indicado finalize.

**Valor de retorno**

void

**Ejemplo de uso**

|  |
| --- |
| #include"maxiwin.h"  usign namespace maxiwin;  int main(){  mensaje(L"Hola, ¡Nos vemos en un segundo!");  // Espera un segundo  espera(1000);  mensaje(L"¡Te habia dicho un segundo!", L"¡HEY!", mensajeria::ICON\_ADVERTENCIA);  return 0;  } |
| Primer mensaje:    ... Espera un segundo...  Segundo mensaje:    ... El programa finaliza. |

**Hacer\_al\_salir**

Firma:

template<typename... Args> void hacer\_al\_salir(Callback<Args...> doing)

**Parametros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***doing*** | Callback<Args…> | Función a ejecutar |  |

**Descripción**

Asigna una función, que se ejecutará cuando se intente cerrar la ventana gráfica.

**Valor de retorno**

void

**Ejemplo de uso**

|  |
| --- |
| #include"maxiwin.h"  usign namespace maxiwin;  void foo(){  mensaje(L"¡Hola, desde la salida!");  }  int main(){  mensaje(L"¡Cierra el programa, ahora mismo!", L"¡Advertencia!", mensajeria::ICON\_ADVERTENCIA);  // Asignar la función  hacer\_al\_salir(foo);  return 0;  } |
| #1:    #2:    #3:    ... El programa finaliza. |

**Cancelar\_cierre**

Firma:

void cancelar\_cierre()

**Descripción**

Le indica a la ventana grafica que no pueda ser cerrada, o por lo menos, durante un intento del usuario.

Su objetivo es ser combinada con funciones como ***maxiwin::hacer\_al\_salir*** y/o ***maxiwin::aceptar\_cierre***.

Cuando el usuario intenté cerrar la ventana, una vez aplicado el efecto de esta función, la acción será negada completamente, y se ejecutará la función asignada por ***maxiwin::hacer\_al\_salir*** (No ejecutará nada, si no se ha asignado).

**Valor de retorno**

void

**Ejemplo de uso**

|  |
| --- |
| #include"maxiwin.h"  usign namespace maxiwin;  int intentos = 0;  void bar(){  intentos++;  if(intentos < 10){  char txt[85]{'\0'};  sprintf(txt, "Error, intentalo nuevamente...\nCantidad de intentos: %d.\nMinimo de intentos: 10.\0", intentos);  // Lo cancelamos una vez más  cancelar\_cierre();  mensaje(txt);  }else if(intentos == 10){  // Permitir el cierre  aceptar\_cierre();  mensaje("Ok, ya puedes irte...");  // Cerrar la ventana  vcierra();  }  }  int main(){  mensaje(L"¡No puedes cerrar este programa, hasta que YO lo diga!");  // Asignar la funcion  hacer\_al\_salir(bar);  // Cancelar el cierre  cancelar\_cierre();  return 0;  } |

**Aceptar\_cierre**

Firma:

void aceptar\_cierre()

**Descripción**

Le indica a la ventana grafica que ya puede ser cerrada, o por lo menos, durante un intento del usuario.

Su objetivo es ser combinada con funciones como ***maxiwin::hacer\_al\_salir*** y/o ***maxiwin::cancelar\_cierre***.

Cuando el usuario intenté cerrar la ventana, una vez aplicado el efecto de esta función, la acción será permitida, por defecto ya esta permitido. Esto sirve muy bien, en caso de que ya hayas negado el acceso con la función ***maxiwin::cancelar\_cierre***.

**Valor de retorno**

void

**Ejemplo de uso**

|  |
| --- |
| #include"maxiwin.h"  usign namespace maxiwin;  int intentos = 0;  void bar(){  intentos++;  if(intentos == 1){  // Permitir el cierre  aceptar\_cierre();  mensaje("Era broma, puedes irte.");  // Cerrar la ventana  vcierra();  }  }  int main(){  mensaje(L"¡No puedes cerrar este programa, hasta que YO lo diga!");  // Asignar la funcion  hacer\_al\_salir(bar);  // Cancelar el cierre  cancelar\_cierre();  return 0;  } |

**Funciones de obtención de entradas**

**Tecla**

Firma:

bool tecla(int key)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***key*** | int | Tecla a detectar |  |

**Descripción**

Comprueba si una tecla fue presionada, por lo menos una vez, teniendo un delay de detección que es inversamente proporcional al tiempo en que dicha tecla está siendo presionada.

Se recomienda consultar con ***maxiwin::teclas***, en caso de que se quiera consultar con teclas de función; como el espacio, flechas direccionales, tecla de retorno, entre otros.

En caso de que se quiera detectar una tecla alfabética, numérica o simbólica, se tendrá que pasar como parámetro el carácter de dicha tecla.

**Valor de retorno**

bool

|  |  |  |
| --- | --- | --- |
| Valor | Condición | Caso |
| ***true*** | … Si … | ***key*** fue presionada. |
| ***false*** | … Si … | ***key*** no fue presionada, o se encuentra en tiempo muerto de detección de dicha tecla. |

**Ejemplo de uso**

|  |
| --- |
| int main(){  while(!tecla('Q')){  if(tecla('W')) mensaje("Arriba");  if(tecla('S')) mensaje("Abajo");  if(tecla('A')) mensaje("Izquierda");  if(tecla('D')) mensaje("Derecha");  }  mensaje("Chau...");  return 0;  } |
| Hasta que no presionemos la tecla ‘Q’:   |  |  | | --- | --- | | ... Tecla ‘W’: | ... Tecla ‘S’: | | ... Tecla ‘A’: | ... Tecla ‘D’: |   ... Una vez presionada la tecla ‘Q’:    ... El programa finaliza. |

**Tecla\_presionada**

Firma:

bool tecla\_presionada(int key)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***key*** | int | Tecla a detectar |  |

**Descripción**

Comprueba si una tecla fue presionada, pero a diferencia de ***maxiwin::tecla***, esta función no posee un delay de detección.

Se recomienda consultar con ***maxiwin::teclas***, en caso de que se quiera consultar con teclas de función; como el espacio, flechas direccionales, tecla de retorno, entre otros.

En caso de que se quiera detectar una tecla alfabética, numérica o simbólica, se tendrá que pasar como parámetro el carácter de dicha tecla.

**Valor de retorno**

|  |  |  |
| --- | --- | --- |
| Valor | Condición | Caso |
| ***true*** | … Si … | ***key*** está siendo presionada. |
| ***false*** | … Si … | ***key*** no está siendo presionada. |

**Raton\_dentro**

Firma:

bool raton\_dentro()

**Descripción**

Comprueba si el puntero del ratón se encuentra dentro o fuera del área de la ventana gráfica.

**Valor de retorno**

bool

|  |  |  |
| --- | --- | --- |
| Valor | Condición | Caso |
| ***true*** | … Si … | El puntero del ratón está dentro de la ventana. |
| ***false*** | … Si … | El puntero del ratón no está dentro de la ventana. |

**Raton\_x**

Firma:

float raton\_x();

**Descripción**

Obtiene la posición en el eje X del puntero del ratón. Solo se actualiza siempre y cuando el puntero del ratón este dentro del área de la ventana.

**Valor de retorno**

float

Proporcional a la posición en pixeles del eje X del puntero del ratón dentro del área de la ventana.

**Raton\_y**

Firma:

float raton\_y();

**Descripción**

Obtiene la posición en el eje Y del puntero del ratón. Solo se actualiza siempre y cuando el puntero del ratón este dentro del área de la ventana.

**Valor de retorno**

float

Proporcional a la posición en pixeles del eje Y del puntero del ratón dentro del área de la ventana.

**Raton**

Firma:

bool raton(float& x, float& y);

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x*** | float& | Numero a modificar para obtener el eje X |  |
| **y** | float& | Numero a modificar para obtener el eje Y |  |

**Descripción**

Cuando el puntero del ratón se encuentra dentro del área de la ventana, retornará ***true*** y modificará el valor de las variables enviadas a ***x*** e ***y***. En caso contrario, retornará ***false***.

**Valor de retorno**

bool

|  |  |  |
| --- | --- | --- |
| Valor | Condición | Caso |
| ***true*** | … Si … | El puntero del ratón está dentro de la ventana. |
| ***false*** | … Si … | El puntero del ratón no está dentro de la ventana. |

**Valores modificados**

|  |  |
| --- | --- |
| Variable involucrada | Valor obtenido |
| ***x*** | Eje X del puntero del ratón |
| ***y*** | Eje Y del puntero del ratón |

**Raton\_boton\_izq**

Firma:

bool raton\_boton\_izq()

**Descripción**

Comprueba si se está presionando el clic izquierdo del ratón.

**Valor de retorno**

bool

|  |  |  |
| --- | --- | --- |
| Valor | Condición | Caso |
| ***true*** | … Si … | Se está presionando el clic izquierdo del ratón |
| ***false*** | … Si … | No se está presionando el clic izquierdo del ratón |

**Raton\_boton\_der**

Firma:

bool raton\_boton\_der()

**Descripción**

Comprueba si se está presionando el clic derecho del ratón.

**Valor de retorno**

bool

|  |  |  |
| --- | --- | --- |
| Valor | Condición | Caso |
| ***true*** | … Si … | Se está presionando el clic derecho del ratón |
| ***false*** | … Si … | No se está presionando el clic derecho del ratón |

**Raton\_boton\_izq\_justo**

Firma:

bool raton\_boton\_izq\_justo()

**Descripción**

Comprueba si se ha presionado el clic izquierdo del ratón, durante un breve periodo de tiempo

**Valor de retorno**

bool

|  |  |  |
| --- | --- | --- |
| Valor | Condición | Caso |
| ***true*** | … Si … | Se ha presionado el clic izquierdo del ratón |
| ***false*** | … Si … | No se ha presionado el clic izquierdo del ratón, o ya se ha detectado |

**Raton\_boton\_der\_justo**

Firma:

bool raton\_boton\_der\_justo()

**Descripción**

Comprueba si se ha presionado el clic derecho del ratón, durante un breve periodo de tiempo

**Valor de retorno**

bool

|  |  |  |
| --- | --- | --- |
| Valor | Condición | Caso |
| ***true*** | … Si … | Se ha presionado el clic derecho del ratón |
| ***false*** | … Si … | No se ha presionado el clic derecho del ratón, o ya se ha detectado |

**Raton\_botones**

Firma:

void raton\_botones(bool& izq, bool& der);

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***izq*** | bool& | Booleano a modificar para obtener el caso del clic izquierdo |  |
| **der** | bool& | Booleano a modificar para obtener el caso del clic derecho |  |

**Descripción**

Obtiene en aluminoso si se está presionando el clic izquierdo o derecho, modificando el valor de las variables enviadas a ***izq*** y ***der***.

**Valor de retorno**

void

**Valores modificados**

|  |  |
| --- | --- |
| Variable involucrada | Valor obtenido |
| ***izq*** | Si se está presionando el clic izquierdo del raton o no. |
| **der** | Si se está presionando el clic derecho del raton o no. |

**Raton\_capturar**

Firma:

void raton\_capturar();

**Descripción**

Captura el puntero del ratón dentro del área visible de la ventana, no funcionará si ya se efectuó esta acción.

**Valor de retorno**

void

**Raton\_** **liberar**

Firma:

void raton\_liberar();

**Descripción**

Libera el puntero del ratón del área visible de la ventana, no funcionará si ya se efectuó esta acción, funciona principalmente para contrarrestar el efecto de ***maxiwin::raton\_capturar***.

**Valor de retorno**

void

**Funciones de control ventanal**

**Pancho**

Firma:

int pancho()

**Descripción**

Obtiene el ancho de la pantalla.

**Valor de retorno**

int

Proporcional al ancho en pixeles de la pantalla.

**Palto**

Firma:

int palto()

**Descripción**

Obtiene el alto de la pantalla.

**Valor de retorno**

int

Proporcional al alto en pixeles de la pantalla.

**Vancho**

Firma:

int vancho()

**Descripción**

Obtiene el ancho de la ventana.

**Valor de retorno**

int

Proporcional al ancho en pixeles de la ventana.

**Valto**

Firma:

int valto()

**Descripción**

Obtiene el alto de la ventana.

**Valor de retorno**

int

Proporcional al alto en pixeles de la ventana.

**Vposicion\_x**

Firma:

int vposicion\_x()

**Descripción**

Obtiene el Eje X de la ventana.

**Valor de retorno**

int

Proporcional al Eje X en pixeles de la ventana.

**Vposicion\_y**

Firma:

int vposicion\_y()

**Descripción**

Obtiene el Eje Y de la ventana.

**Valor de retorno**

int

Proporcional al Eje Y en pixeles de la ventana.

**Vtitulo**

Firma:

DynArray<char> vtitulo()

**Descripción**

Obtiene el titulo de la ventana.

**Valor de retorno**

DynArray<char>

Proporcional al titulo de la ventana.

**V\_fijar\_titulo**

Firma:

void v\_fijar\_titulo(const DynArray<char> title)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***title*** | DynArray<char> | Titulo a establecer |  |

**Descripción**

Establece el titulo en la ventana al de ***title***.

**Valor de retorno**

void

**Vredimensiona**

Firma:

void vredimensiona(int ample, int alt)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***ample*** | int | Nuevo ancho |  |
| ***alt*** | int | Nuevo alto |  |

**Descripción**

Establece el ancho y alto de la ventana a los valores de ***ample*** y ***alt*** respectivamente.

**Valor de retorno**

void

**Vmover**

Firma:

void vmover(int x, int y)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x*** | int | Nuevo Eje X |  |
| ***y*** | int | Nuevo Eje Y |  |

**Descripción**

Establece la posición en ejes X e Y de la ventana, a los valores de ***x*** e ***y*** respectivamente.

**Valor de retorno**

void

**Vcentrar**

Firma:

void vcentrar()

**Descripción**

Establece la posición en ejes X e Y de la ventana, al centro de la pantalla, centrando la ventana en proporción al tamaño de esta misma.

**Valor de retorno**

void

**Vminimiza**

Firma:

void vminimiza()

**Descripción**

Minimiza la ventana.

**Valor de retorno**

void

**Vmaximiza**

Firma:

void vmaximiza()

**Descripción**

Maximiza la ventana.

**Valor de retorno**

void

**Vcompleta**

Firma:

void vcompleta()

**Descripción**

Maximiza la ventana, de tal forma que ocupe toda el área de la pantalla.

**Valor de retorno**

void

**Vicono**

Firma:

img vicono()

**Descripción**

Obtiene el icono actual de la ventana.

**Valor de retorno**

img

Proporcional al icono actual de la ventana.

**V\_fijar\_icono**

Firma:

void v\_fijar\_icono(const img& hImage)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***hImage*** | const img& | Imagen a establecer |  |

**Descripción**

Establece el icono actual de la ventana.

**Valor de retorno**

void

**Voculta**

Firma:

void voculta()

**Descripción**

Establece la visualización de la ventana como invisible, esto también causando que el área de dibujado también lo sea.

**Valor de retorno**

void

**Vmuestra**

Firma:

void vmuestra()

**Descripción**

Establece la visualización de la ventana como visible, esto también causando que el área de dibujado también lo sea. Siendo el opuesto de ***maxiwin::voculta***, que contrarresta sus efectos.

**Valor de retorno**

void

**Vcierra**

Firma:

void vcierra()

**Descripción**

Cierra la ventana.

Puede ser cancelado por la función ***maxiwin::cancelar\_cierre***.

Antes de ejecutar la acción para cerrar la ventana, se ejecuta la función asignada por ***maxiwin::hacer\_al\_salir*** (Si se le fue asignada), independientemente si ***maxiwin::cancelar\_cierre*** efectuó su efecto.

**Valor de retorno**

void

**Funciones de contexto gráfico**

**Color**

Firma:

uint32\_t color()

**Descripción**

Obtiene el color actual del contexto.

**Valor de retorno**

uint32\_t

Proporcional al color del contexto gráfico. Por defecto, el valor es ***0xFFFFFFFF***.

**Fijar\_color**

Firma:

void fijar\_color(const uint32\_t& \_c)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***\_c*** | const uint32\_t& | Color a establecer |  |

**Descripción**

Establece el color del contexto gráfico actual al valor de ***\_c***.

**Valor de retorno**

void

**Lapiz**

Firma:

float lapiz()

**Descripción**

Obtiene el grosor del lápiz actual del contexto.

**Valor de retorno**

float

Proporcional al grosor de lápiz del contexto actual. Por defecto, el valor es ***1.0f***.

**Fijar\_lapiz**

Firma:

void fijar\_lapiz(float size = 1.0f)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***size*** | float | Grosor de lapiz a establecer | 1.0f |

**Descripción**

Establece el grosor de lápiz del contexto gráfico actual al valor de ***size***, siendo el valor mínimo de ***1.0f***.

**Valor de retorno**

void

**Camara**

Firma:

cam camara()

**Descripción**

Obtiene las configuraciones de cámara actual del contexto.

**Valor de retorno**

cam

Proporcional a las configuraciones de cámara 3D del contexto actual.

**Fijar\_camara**

Firma:

void fijar\_camara(const cam& \_c)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***\_c*** | const cam& | Configuraciones de cámara a establecer |  |

**Descripción**

Establece las configuraciones de cámara del contexto gráfico actual al de ***\_c***.

**Valor de retorno**

void

**Fuente**

Firma:

fnt fuente()

**Descripción**

Obtiene la fuente actual del contexto.

**Valor de retorno**

cam

Proporcional la fuente del contexto actual.

**Fijar\_fuente**

Firma:

void fijar\_fuente(const fnt& fuente)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***fuente*** | const fnt& | Fuente a establecer |  |

**Descripción**

Establece la fuente del contexto gráfico actual a ***fuente***.

**Valor de retorno**

void

Firma:

void fijar\_fuente(const DynArray<char>& fuente, unsigned int dimension = 12, unsigned int \_styles = 0)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***fuente*** | const DynArray<char>& | Nombre de la fuente a establecer |  |
| ***dimension*** | unsigned int | Tamaño de la fuente | 12 |
| ***\_styles*** | unsigned int | Estilo de la fuente | 0 |

**Descripción**

Crea un nuevo ***fnt***, a partir de ***fuente***, ***dimension*** y ***\_styles***; para inmediatamente establecerlo al contexto gráfico actual.

Se recomienda consultar con ***maxiwin::fuente\_estilos*** para el valor de ***\_styles***.

**Valor de retorno**

void

**Obtener\_fuentes**

Firma:

DynArray<DynArray<char>> obtener\_fuentes()

**Descripción**

Obtiene todas las fuentes añadidas al contexto gráfico, sumando las por defecto del sistema inclusive.

**Valor de retorno**

DynArray<DynArray<char>>

Lista de textos correspondientes a cada nombre de las fuentes asociadas al contexto grafico actual.

Tenga en cuenta que estas funciones influyen en todas que pertenezcan a la categoría de Funciones de dibujo primitivo.

**Funciones de dibujo primitivo**

Cabe aclarar que toda función de dibujo primitivo tendrá como valor de retorno ***void***.

**Borra**

Firma:

void borra()

**Descripción**

Borra todo el contenido gráfico dibujado previamente, no hay excepciones.

**Refresca**

Firma:

void refresca()

**Descripción**

Muestra todo el contenido gráfico dibujado sobre la ventana, no hay excepciones.

Todo contenido gráfico dibujado no se mostrará sobre la ventana hasta que se ejecute esta función, no hay excepciones.

**Punto**

Firma:

void punto(float x, float y)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x*** | float | Eje X |  |
| ***y*** | float | Eje Y |  |

**Descripción**

Dibuja un pixel en las coordenadas especificadas por ***x*** e ***y***.

Su color es proporcional al del contexto gráfico actual.

**Balde**

Firma:

void balde(float x, float y)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x*** | float | Eje X |  |
| ***y*** | float | Eje Y |  |

**Descripción**

Realiza un llenado de balde, empezando a partir de las coordenadas ***x*** e ***y***, a medida que va encontrando el mismo color dentro del contexto gráfico, el único requisito es que cuyo color encontrado conecte con el anterior en sus lados ortogonales adyacentes. Parará cuando logré cubrir todo el contexto gráfico, o cuando ya no hallan coincidencias de color.

Cabe aclarar que el algoritmo de esta función es de DPS, por lo que no se garantiza que se logré pintar con esta función en un tiempo pseudo linealmente corto.

Su color es proporcional al del contexto gráfico actual.

**Linea**

Firma:

void linea(float x\_ini, float y\_ini, float x\_fin, float y\_fin)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x\_ini*** | float | Eje X inicial |  |
| ***y\_ini*** | float | Eje Y inicial |  |
| ***x\_fin*** | float | Eje X final |  |
| ***y\_fin*** | float | Eje Y final |  |

**Descripción**

Dibuja una línea desde las coordenadas iniciales (***x\_ini*** y ***y\_ini***) hasta las coordenadas finales (***x\_fin*** y ***y\_fin***).

Su color es proporcional al del contexto gráfico actual.

Su grosor es proporcional al grosor de lápiz del contexto gráfico actual.

**Rectangulo**

Firma:

void rectangulo(float izq, float arr, float der, float aba)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***izq*** | float | Eje X inicial |  |
| ***arr*** | float | Eje Y inicial |  |
| ***der*** | float | Eje X final |  |
| ***aba*** | float | Eje Y final |  |

**Descripción**

Dibuja el perímetro de un rectángulo formado por el área creada a partir de las coordenadas iniciales (***izq*** y ***arr***) hasta las coordenadas finales (***der*** y ***aba***).

El color de ese perímetro es proporcional al del contexto gráfico actual.

El grosor de ese perímetro es proporcional al grosor de lápiz del contexto gráfico actual.

**Rectangulo\_lleno**

Firma:

void rectangulo\_lleno(float izq, float arr, float der, float aba)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***izq*** | float | Eje X inicial |  |
| ***arr*** | float | Eje Y inicial |  |
| ***der*** | float | Eje X final |  |
| ***aba*** | float | Eje Y final |  |

**Descripción**

Dibuja el área de un rectángulo formado partir de las coordenadas iniciales (***izq*** y ***arr***) hasta las coordenadas finales (***der*** y ***aba***).

El color de esa área es proporcional al del contexto gráfico actual.

**Circulo**

Firma:

void circulo(float x\_cen, float y\_cen, float radio)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x\_cen*** | float | Eje X central |  |
| ***y\_cen*** | float | Eje Y central |  |
| ***radio*** | float | Radio del circulo |  |

**Descripción**

Dibuja el perímetro de un circulo formado por su centro (***x\_cen*** y ***y\_cen***) y su radio (***radio***).

El color de ese perímetro es proporcional al del contexto gráfico actual.

El grosor de ese perímetro es proporcional al grosor de lápiz del contexto gráfico actual.

**Circulo** **\_lleno**

Firma:

void circulo\_lleno(float x\_cen, float y\_cen, float radio)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x\_cen*** | float | Eje X central |  |
| ***y\_cen*** | float | Eje Y central |  |
| ***radio*** | float | Radio del circulo |  |

**Descripción**

Dibuja el área de un circulo formado por su centro (***x\_cen*** y ***y\_cen***) y su radio (***radio***).

El color de esa área es proporcional al del contexto gráfico actual.

**Elipse**

Firma:

void elipse(float x\_cen, float y\_cen, float radio, float anguloInicio, float anguloFinal)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x\_cen*** | float | Eje X central |  |
| ***y\_cen*** | float | Eje Y central |  |
| ***radio*** | float | Radio del circulo |  |
| ***anguloInicio*** | float | Angulo de inicio del dibujado |  |
| ***anguloFinal*** | float | Angulo de final del dibujado |  |

**Descripción**

Dibuja el área correspondiente al perímetro de un circulo, dicho circulo siendo formado por su coordenada central (***x\_cen*** y ***y\_cen***) y su radio (***radio***), siendo que el área es delimitada por el angulo inicial (***anguloInicio***) y el final (anguloFinal); dichos angulos moviéndose en sentido antihorario e iniciados apuntando hacia el este.

El color de esa área es proporcional al del contexto gráfico actual.

El grosor de esa área es proporcional al grosor de lápiz del contexto gráfico actual.

**Elipse\_lleno**

Firma:

void elipse\_lleno(float x\_cen, float y\_cen, float radio, float anguloInicio, float anguloFinal)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x\_cen*** | float | Eje X central |  |
| ***y\_cen*** | float | Eje Y central |  |
| ***radio*** | float | Radio del circulo |  |
| ***anguloInicio*** | float | Angulo de inicio del dibujado |  |
| ***anguloFinal*** | float | Angulo de final del dibujado |  |

**Descripción**

Dibuja el área correspondiente al área de un circulo, dicho circulo siendo formado por su coordenada central (***x\_cen*** y ***y\_cen***) y su radio (***radio***), siendo que el área es delimitada por el angulo inicial (***anguloInicio***) y el final (anguloFinal); dichos angulos moviéndose en sentido antihorario e iniciados apuntando hacia el este.

El color de esa área es proporcional al del contexto gráfico actual.

**Arco**

Firma:

void arco(float x1, float y1, float x2, float y2, float x3, float y3)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x1*** | float | Eje X de la coordenada A |  |
| ***y1*** | float | Eje Y de la coordenada A |  |
| ***x2*** | float | Eje X de la coordenada B |  |
| ***y2*** | float | Eje Y de la coordenada B |  |
| ***x3*** | float | Eje X de la coordenada C |  |
| ***y3*** | float | Eje Y de la coordenada C |  |

**Descripción**

Dibuja el perímetro un arco, que inicia desde la coordenada A (***x1*** y ***y1***), pasa por B (***x2*** y ***y2***) y termina en C (***x3*** y ***y3***).

El color de ese perímetro es proporcional al del contexto gráfico actual.

El grosor de ese perímetro es proporcional al grosor de lápiz del contexto gráfico actual.

**Arco\_lleno**

Firma:

void arco\_lleno(float x1, float y1, float x2, float y2, float x3, float y3)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x1*** | float | Eje X de la coordenada A |  |
| ***y1*** | float | Eje Y de la coordenada A |  |
| ***x2*** | float | Eje X de la coordenada B |  |
| ***y2*** | float | Eje Y de la coordenada B |  |
| ***x3*** | float | Eje X de la coordenada C |  |
| ***y3*** | float | Eje Y de la coordenada C |  |

**Descripción**

Dibuja el área un arco, que inicia desde la coordenada A (***x1*** y ***y1***), pasa por B (***x2*** y ***y2***) y termina en C (***x3*** y ***y3***).

El color de esa área es proporcional al del contexto gráfico actual.

**Triangulo**

Firma:

void triangulo(float x1, float y1, float x2, float y2, float x3, float y3)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x1*** | float | Eje X de la coordenada A |  |
| ***y1*** | float | Eje Y de la coordenada A |  |
| ***x2*** | float | Eje X de la coordenada B |  |
| ***y2*** | float | Eje Y de la coordenada B |  |
| ***x3*** | float | Eje X de la coordenada C |  |
| ***y3*** | float | Eje Y de la coordenada C |  |

**Descripción**

Dibuja el perímetro de un triangulo, formado por sus coordenadas A (***x1*** y ***y1***), B (***x2*** y ***y2***) y C (***x3*** y ***y3***).

El color de ese perímetro es proporcional al del contexto gráfico actual.

El grosor de ese perímetro es proporcional al grosor de lápiz del contexto gráfico actual.

**Triangulo\_lleno**

Firma:

void triangulo\_lleno(float x1, float y1, float x2, float y2, float x3, float y3)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x1*** | float | Eje X de la coordenada A |  |
| ***y1*** | float | Eje Y de la coordenada A |  |
| ***x2*** | float | Eje X de la coordenada B |  |
| ***y2*** | float | Eje Y de la coordenada B |  |
| ***x3*** | float | Eje X de la coordenada C |  |
| ***y3*** | float | Eje Y de la coordenada C |  |

**Descripción**

Dibuja el área de un triangulo, formado por sus coordenadas A (***x1*** y ***y1***), B (***x2*** y ***y2***) y C (***x3*** y ***y3***).

El color de esa área es proporcional al del contexto gráfico actual.

**Poligono**

Firma:

void poligono(const DynArray<float>& \_x, const DynArray<float>& \_y)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***\_x*** | const DynArray<float>& | Lista de Ejes X |  |
| ***\_y*** | const DynArray<float>& | Lista de Ejes Y |  |

**Descripción**

Dibuja el perímetro de un polígono cerrado con una cantidad de lados y coordenadas proporcional al de las listas brindadas (***\_x*** y ***\_y***).

El perímetro no se dibujará en caso de que las dos listas pasadas a ***\_x*** y ***\_y*** tengan tamaños desiguales, o menores a 2.

El color de ese perímetro es proporcional al del contexto gráfico actual.

El grosor de ese perímetro es proporcional al grosor de lápiz del contexto gráfico actual.

**Poligono\_lleno**

Firma:

void poligono(const DynArray<float>& \_x, const DynArray<float>& \_y)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***\_x*** | const DynArray<float>& | Lista de Ejes X |  |
| ***\_y*** | const DynArray<float>& | Lista de Ejes Y |  |

**Descripción**

Dibuja el área de un polígono con una cantidad de lados y coordenadas proporcional al de las listas brindadas (***\_x*** y ***\_y***).

El área no se dibujará en caso de que las dos listas pasadas a ***\_x*** y ***\_y*** tengan tamaños desiguales, o menores a 2.

El color de esa área es proporcional al del contexto gráfico actual.

**Texto**

Firma:

void texto(float x, float y, const DynArray<char>& txt)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x*** | float | Eje X inicial |  |
| ***y*** | float | Eje Y inicial |  |
| ***txt*** | const DynArray<char>& | Texto a imprimir |  |

**Descripción**

Escribe el texto proporcionado a ***txt***, iniciando desde la coordenada formada por ***x*** y ***y***.

El color del texto es proporcional al del contexto gráfico actual.

La tipografía del texto es proporcional a la configuración de fuente del contexto gráfico actual.

**Funciones de dibujo avanzado**

**Imagen**

Firma:

void imagen(float x, float y, const DynArray<char>& filePath)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x*** | float | Eje X |  |
| ***y*** | float | Eje Y |  |
| ***filePath*** | const DynArray<char>& | Dirección del directorio |  |

**Descripción**

Variación de esta función que permite el dibujado instantáneo de una imagen desde un directorio, archivo o carpeta del computador (la ruta pasada a ***filePath***); dibujándola a partir de la coordenada formada por ***x*** e ***y***, dibujándola desde si esquina superior izquierda.

**Valor de retorno**

void

Firma:

void imagen(float x, float y, const img& hImage)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x*** | float | Eje X |  |
| ***y*** | float | Eje Y |  |
| ***hImage*** | const img& | Imagen a dibujar |  |

**Descripción**

Variación de esta función que permite dibujar una imagen que el programa halla almacenado en memoria, dibujándola a partir de la coordenada formada por ***x*** e ***y***, dibujándola desde su esquina superior izquierda.

**Valor de retorno**

void

Firma:

void imagen(float x, float y, anim& hAnimation);

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x*** | float | Eje X |  |
| ***y*** | float | Eje Y |  |
| ***hAnimation*** | anim& | Animación a dibujar |  |

**Descripción**

Variación de esta función que permite dibujar una animación que el programa halla almacenado en memoria, dibujándola a partir de la coordenada formada por ***x*** e ***y***, dibujándola desde su esquina superior izquierda.

Una vez dibujada, la animación avanzará automáticamente un frame.

**Valor de retorno**

void

Firma:

void imagen(const pnt& hPoint);

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***hPoint*** | const pnt& | Punto a dibujar |  |

**Descripción**

Variación de esta función que permite dibujar un punto que el programa halla almacenado en memoria, dibujándolo a partir de las coordenadas proporcionadas por el punto.

Firma:

void imagen(float x, float y, const mdl& hModel)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***x*** | float | Eje X 3D |  |
| ***y*** | float | Eje Y 3D |  |
| ***hModel*** | const mdl& | Modelo 3D a dibujar |  |

**Descripción**

Variación de esta función que permite dibujar un modelo 3D que el programa halla almacenado en memoria, dibujándola a partir del pivote de dicho modelo, siendo ***x*** e ***y*** una suma extra para la posición del pivote.

**Valor de retorno**

void

Firma:

void imagen(const mdl& hModel);

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***hModel*** | const mdl& | Modelo 3D a dibujar |  |

**Descripción**

Variación de esta función que permite dibujar un modelo 3D que el programa halla almacenado en memoria, dibujándola a partir del pivote de dicho modelo.

**Valor de retorno**

void

**Cargar\_imagen**

Firma:

img cargar\_imagen(const DynArray<char>& filePath)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***filePath*** | const DynArray<char>& | Dirección del directorio |  |

**Descripción**

Función que permite cargar una imagen desde un directorio, archivo o carpeta del computador (la ruta pasada a ***filePath***).

**Valor de retorno**

img

Proporcional a los datos de la imagen traducidos a un formato semejante a un BitMap.

**Cargar\_animacion**

Firma:

anim cargar\_animacion(const DynArray<char>& gifPath)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***gifPath*** | const DynArray<char>& | Dirección del directorio |  |

**Descripción**

Función que permite cargar una animación desde un directorio, archivo o carpeta del computador (la ruta pasada a ***gifPath***).

**Valor de retorno**

img

Proporcional a los datos de la animación traducidos a un formato semejante a un conjunto de BitMaps.

**Ancho\_imagen** y **Alto\_imagen**

Firmas:

int ancho\_imagen(const DynArray<char>& filePath)

int alto\_imagen(const DynArray<char>& filePath)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***filePath*** | const DynArray<char>& | Dirección del directorio |  |

**Descripción**

Ambas permiten poder obtener de forma instantánea el ancho y/o alto de una imagen almacenada en un directorio, archivo o carpeta del equipo, sin necesidad de cargarla.

**Valor de retorno de ambas funciones**

int

Proporcional a el ancho y alto de la imagen respectivamente.

**Cargar\_fuente**

Firma:

fnt cargar\_fuente(const DynArray<char>& filePath)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***filePath*** | const DynArray<char>& | Dirección del directorio |  |

**Descripción**

Permite poder cargar una fuente de texto desde un directorio, archivo o carpeta del computador (la ruta pasada a ***filePath***).

**Valor de retorno**

fnt

Proporcional a las configuraciones de esta fuente.

**Cargar\_modelo**

Firma:

mdl cargar\_modelo(const DynArray<char>& filePath, double scale = 1.0, bool load\_textures = true)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***filePath*** | const DynArray<char>& | Dirección del directorio |  |
| ***scale*** | double | Escala del modelo | ***1.0*** |
| ***load\_textures*** | bool | “¿Quiere cargar las texturas del modelo?” | ***true*** |

**Descripción**

Permite poder cargar un modelo 3D de texto desde un directorio, archivo o carpeta del computador (la ruta pasada a ***filePath***).

El tamaño final del modelo, dependerá del escalar ***scale***.

El que el modelo tenga texturas o no, será infuenciado por ***load\_textures***.

**Valor de retorno**

mdl

Proporcional a los datos del modelo 3D.

**Captura\_de\_ventana**

Firma:

img captura\_de\_ventana()

**Descripción**

Obtiene todo el contenido gráfico dibujado previamente, guardando una copia del mismo en una imagen, no hay excepciones.

Esta función aplica incluso sin necesidad de llamar previamente a ***maxiwin::refresca***, puesto que esta función solo le interesa el contenido gráfico dibujado previantente, no el contenido que se este mostrando en la ventana.

Sirve muy bien para hacer cosas como capturas de pantalla, gráficos prerenderizados, shaders, etc.

**Valor de retorno**

img

Un duplicado que es proporcional al contenido dibujado sobre el contexto gráfico.

**Lienzo**

Firma:

template<typename Func, typename... Args> img& lienzo(img& hImage, Func fn, Args&&... args)

**Parámetros**

|  |  |  |  |
| --- | --- | --- | --- |
| Nombre | Tipo de dato | Función | Valor por defecto |
| ***hImage*** | img& | Imagen a modificar |  |
| ***fn*** | Función de cualquier tipo | Función que hace referencia al procedimiento a aplicar sobre la ***hImage***. |  |
| ***args…*** | Parámetros empaquetados | Parámetros de la función eviada en ***fn***. | ***…*** |

**Descripción**

Aplica un procedimiento grafico (Sea primitivo o complejo) sobre la imagen ***hImage***.

La cantidad de argumentos que hay que pasarle a ***args*** es proporcional a los de la función pasada en ***fn***.

Los efectos graficos no serán aplicados si hubo un error.

Siempre y cuando los tipos de los argumentos sean compatibles con los de la función original, se pueden mezclar, aplica para toda función pasada a ***fn***, y sin excepciones.

No se garantiza que todas las funciones de **Maxiwin**estén registradas dentro del funcionamiento de esta función.

**Valor de retorno**

img&

Proporcional a la referencia a la imagen origen pasada a ***hImage***.

**Ejemplo de uso**

|  |
| --- |
| int main(){  // Creamos una imagen de 50x50 pixeles  img canvas(50, 50);  // Dibujar un pixel azul sobre la imagen  fijar\_color(RGBA(0,0,255,255));  lienzo(canvas, punto, 30, 5);  // Dibujar una linea roja sobre la imagen  fijar\_color(RGBA(255,0,0,255));  lienzo(canvas, linea, 32.0f, 49.0f, 49, 0);  // Dibujar un rectangulo verde sobre la imagen  fijar\_color(RGBA(0,255,0,255));  lienzo(canvas, rectangulo, 32, 49, 49, 0);  // Dibujar un poligono morado sobre la imagen  fijar\_color(RGBA(255,0,255,255));  lienzo(canvas, poligono\_lleno, {0,0,30}, {0,25,45});  // Muestra los resultados  imagen(0,0,canvas);  refresca();  return 0;  } |
|  |

Firma:

template<typename... Args> img& lienzo(img& hImage, \_MAXIWIN\_H\_\_imagen\_, Args&&... args)

Subfirma de automatización, para diferenciar entre ***maxiwin::imagen*** de el resto de funciones registradas de ***maxiwin::lienzo***.

Firmas:

img& lienzo(img& hImage, \_MAXIWIN\_H\_\_poligono\_lleno\_, const DynArray<float> a1, const DynArray<float> a2)

template<typename... Args> img& lienzo(img& hImage, \_MAXIWIN\_H\_\_poligono\_lleno\_, const float(&\_x)[N],const float(&\_y)[N]){return lienzo(hImage, poligono\_lleno, DynArray<float>(\_x), DynArray<float>(\_y));}

Subfirma de automatización, para diferenciar entre ***maxiwin::polígono\_lleno*** de el resto de funciones registradas de ***maxiwin::lienzo***.