

# **VOLMARK**

**Software Package for Digital 3D Image  
(Volume) Copyright Protection**

**v.1.2**



**Thessaloniki 2002**



## 0. Contents

<b>0. Contents</b> .....	<b>3</b>
<b>1. Introduction.</b> .....	<b>5</b>
<b>2. Features.</b> .....	<b>5</b>
<b>3. Volmark DLL Module v.1.2</b> .....	<b>5</b>
3.1 Embedding. ....	5
3.2 Basic Detection. ....	6
3.3 Exhaustive Detection. ....	6
3.4 Parameters. ....	7
3.5 System Requirements. ....	10
<b>4. Volmark SDK Library v.1.2</b> .....	<b>10</b>
4.1 Description. ....	10
4.2 Use. ....	10
4.3 Definitions of SDK Library Functions.....	10
4.3.1 Embedding 3D Watermarks. ....	10
4.3.2 Computing 3D Pick SNR Value. ....	11
4.3.3 Detecting 3D Watermarks (basic detection).....	12
4.3.4 Detecting 3D Watermarks (exhaustive detection).....	13
4.4 Sample Code. ....	14



## 1. Introduction.

**Volmark** is a powerful and flexible software package for embedding and detecting 3D watermarks (signatures) on digital, grayscale, 3D images (volumes). It can be used for copyright protection of 3D images. A copyright owner can embed his unique, personal key number as a 3D watermark in his digital 3D images. The results of 3D watermark detection can be used either for security applications or as legal proof.

## 2. Features.

**Volmark** is based on a blind method for 3D image watermarking robust against geometric distortions. Construction of the 3D watermark having appropriate structure enables fast and robust 3D watermark detection even after several geometric distortions of the watermarked volume. The method used is also robust against lossy compression up to a certain compression ratio.

**Volmark** is distributed in two forms: **Volmark DLL Module v.1.2**, which is a Win32 Dynamic Link Library implemented in C with partial use of C++ and MFC and **Volmark SDK Library v.1.2**, which is a Win32 Static Library implemented exclusively in C. Detailed descriptions of both forms are presented in the following sections (actually, there is a third form of **Volmark**, **Volmark Library v.1.2**, not presented in this manual).

## 3. Volmark DLL Module v.1.2

The **Volmark** module is included in the file `VOLMARK.DLL`. If this DLL exists in the directory of EIKONA3D for Windows when the program starts, a sub-menu called **Volmark** is added under the **Modules** menu. The menu options of this sub-menu provide the necessary tools for 3D Image Watermarking.

The **Volmark** sub-menu provides three options to the user: **Embedding**, **Basic Detection** and **Exhaustive Detection**. Every option is applied on a grayscale volume, which must be loaded in advance through the **File** → **Open** menu. Each option is described below.

### 3.1 Embedding.

This option performs the embedding of a 3D watermark in a given grayscale volume.

The user first selects the volume in which the watermark will be embedded, through the *Select Volume to be Watermarked* dialog box. Then, the user specifies the parameters of the watermark, which will be generated and embedded in the previously selected volume, using the *Embedding Parameters* dialog box (Figure 1). These parameters are: The

*Personal Key Number* used to generate a unique, key-dependent watermark structure and the *Watermark Power Level* used to specify the strength of the generated watermark against attacks (higher level → more robust). In this version of **Volmark DLL Module**, the *Personal Key Number* has a fixed value, determined by the module distributor. Next, the user specifies the output volume, where the watermarked volume will be stored, through the *Select Output Volume* dialog box. Then, the module generates and embeds the watermark in the selected volume. Finally, an info box informs the user of the procedure's issue and the distortion inserted in the watermarked volume. The watermarked volume is placed in the selected output volume buffer and this buffer is displayed.

### 3.2 Basic Detection.

This option performs a basic (fast but not so robust to attacks) search for 3D watermark existence in a given grayscale volume.

The user first selects the volume, which is going to be examined for watermark existence, through the *Select Volume to be Searched* dialog box. Then, the user specifies the *Personal Key Number*, which is supposed to have been used for watermarking the selected volume, using the *Detection Parameters* dialog box (Figure 3). Next, the module generates and searches for the watermark in the selected volume. Finally, an info box informs the user of the procedure's issue. The result of the procedure is a binary answer to the question: "Is the selected volume watermarked with the given personal key number?". The decision is taken by comparing the watermark detector's output (*Detection Ratio*) with a given *Detection Ratio Threshold*.

### 3.3 Exhaustive Detection.

This option performs an exhaustive, thorough search for 3D watermark existence in a given grayscale volume. This procedure is slower than the previous one but enables great robustness against geometric distortions of the given watermarked volume.

The user first selects the volume, which is going to be examined for watermark existence, through the *Select Volume to be Searched* dialog box. Then, the user enters the *Personal Key Number*, which is supposed to have been used for watermarking the selected volume. The user also specifies the scaling range for which he wants the selected volume to be examined for watermark existence. Next, the module generates and searches for the watermark in successive scaled versions of the selected volume in the scaling factor range defined previously. These parameters are specified using the *Detection Parameters* dialog box (Figure 4). Finally, an info box informs the user of the procedure's issue. The result of the procedure is a binary answer as in the previous procedure.

### 3.4 Parameters.

All the watermarking parameters, mentioned above, appear predefined in the parameter dialog boxes. The user can define these values by altering the content of VOLMARK.PAR text file. This file accompanies the VOLMARK.DLL and must be situated in the EIKONA3D directory. In VOLMARK.PAR file the user can predefine the *Personal Key Number* (to be used only for the detection procedures), the *Watermark Power Level* he is willing to use and also the *Watermark Detection Threshold* used in taking decision for watermark existence in a volume. If VOLMARK.PAR does not exist in the directory of EIKONA3D, default values, defined by the module, appear in the watermarking parameter windows. In order to proceed properly, all the parameters inserted in the module (manually or through VOLMARK.PAR) are checked for belonging in the expected value sets, if not: default values are used and the user is informed respectively. These value sets are the following:

<i>Personal Key Number:</i>	integer.
<i>Watermark Power Level:</i>	1, 2, 3 and 4.
<i>Detection Ratio Threshold:</i>	0.5 → 1.0.
<i>Scaling Factors:</i>	0.5 → 1.2.

Note that if a **demo** version of **Volmark DLL Module** is used, the *Personal Key Number* can be defined by the user (both for embedding and detecting 3D watermarks), but is restricted to the following value set:

<i>Personal Key Number:</i>	100000 → 100100.
-----------------------------	------------------

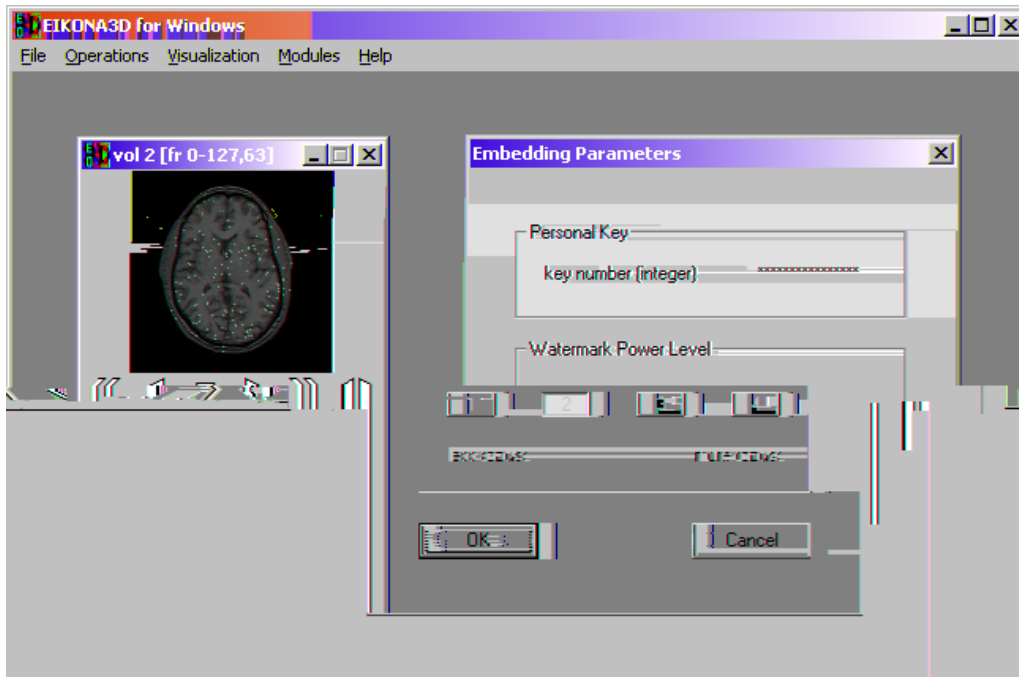


Figure 1: The *Embedding Parameters* dialog box.

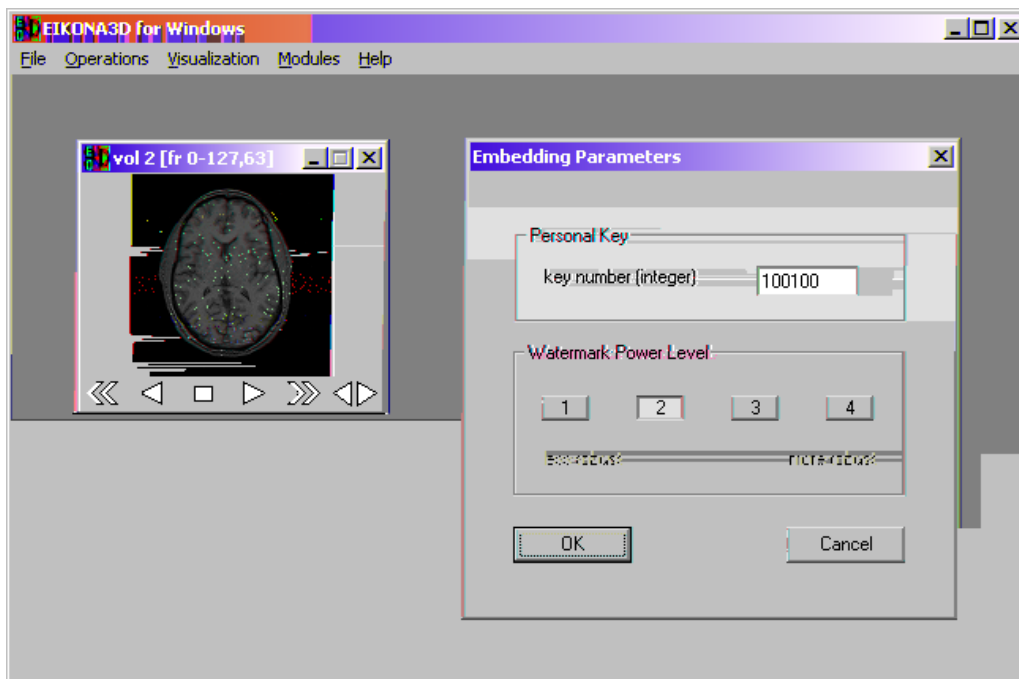
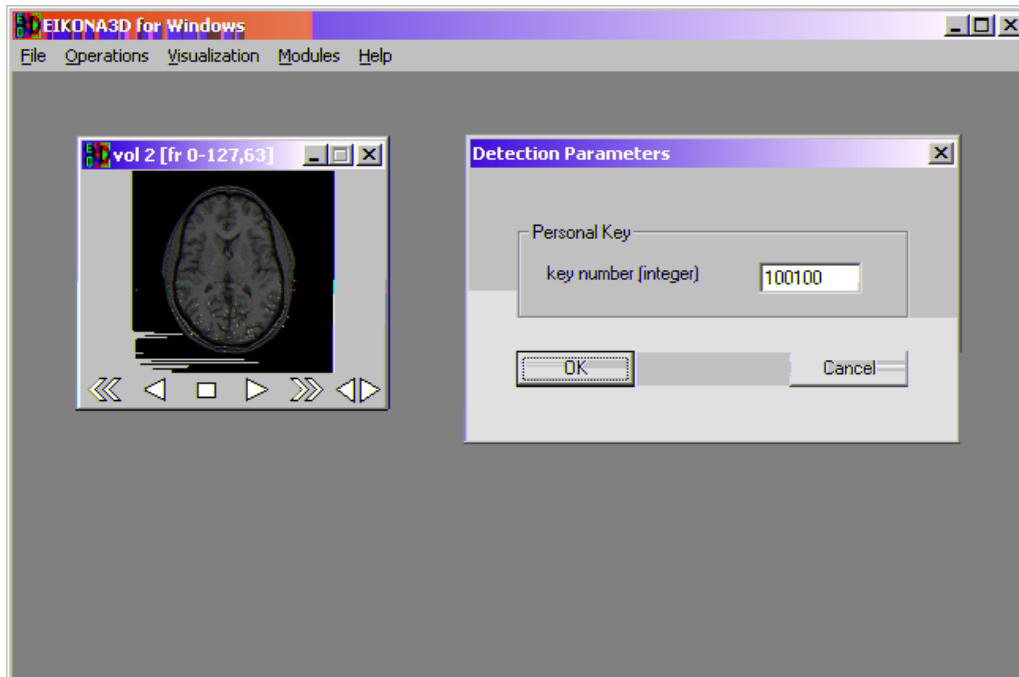
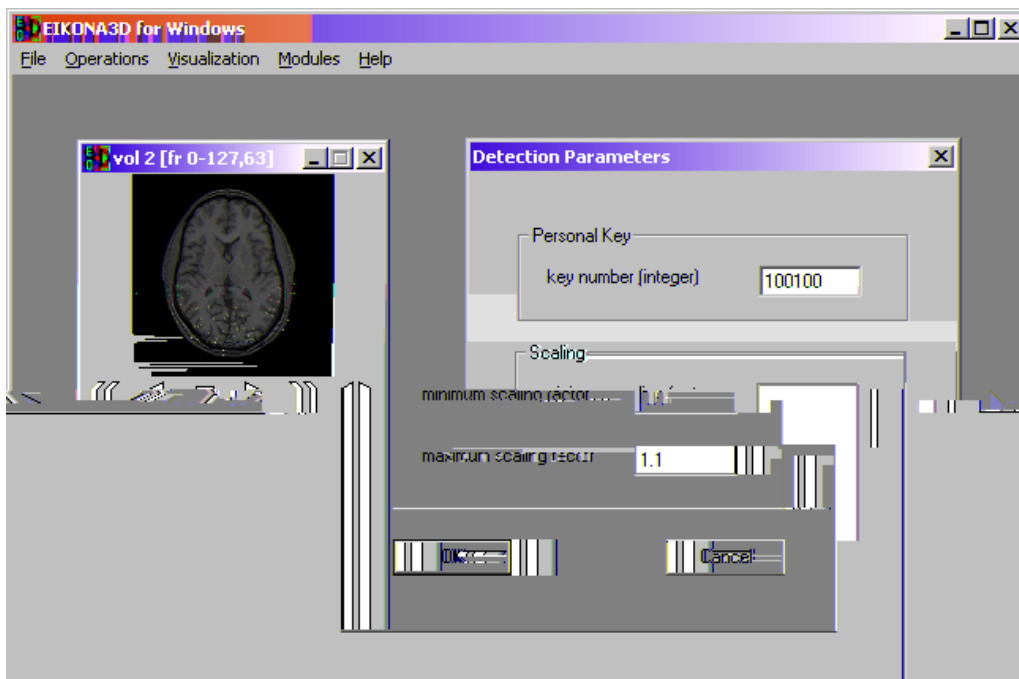


Figure 2: The *Embedding Parameters* dialog box (demo version).



**Figure 3:** The *Detection Parameters* dialog box of the *Basic Detection* option.



**Figure 4:** The *Detection Parameters* dialog box of the *Exhaustive Detection* option.

### 3.5 System Requirements.

The **Volmark DLL Module** works under Eikona3D software, as mentioned above. Therefore, Eikona3D should be installed on your system in order to use the module. Further information about Eikona3D software can be found in the following URL address: [www.alphatecltd.com](http://www.alphatecltd.com).

## 4. Volmark SDK Library v.1.2

### 4.1 Description.

This SDK library provides the necessary functions for copyright protection of grayscale 3D images (volumes). The functions included enable: embedding and detection of 3D watermarks and also computation of 3D pick SNR value.

### 4.2 Use.

The package comes in 2 files, the `VOLMARK_SDK.LIB` (the SDK library itself) and the `VOLMARK_SDK.H`, a header file containing the definitions of the functions included in the SDK library file. The `VOLMARK_SDK.LIB` is a static Win32 library; therefore it must be statically linked to the application project under development, for the functions to be used.

### 4.3 Definitions of SDK Library Functions.

Note that all the functions contained in the SDK library are fully functional, with one restriction: The key used to generate the unique, key-dependent 3D watermark structure is fixed and cannot be altered by the user of the library.

#### 4.3.1 Embedding 3D Watermarks.

The `embedding_sdk()` function embeds a 3D watermark into a grayscale volume ( $v$ ) and thus the watermarked volume ( $v_w$ ) is produced.

```
int embedding_sdk      (   unsigned char ***V, unsigned char ***Vw,
                        int len, int wid, int hei,
                        int plevel                                )
```

#### Input:

- `unsigned char ***V:` a pointer to a 3D array containing the original volume to be watermarked.
- `int len:` the length (in voxels) of the original volume.
- `int wid:` the width (in voxels) of the original volume.
- `int hei:` the height (in voxels) of the original volume.

- `int plevel:` the power level used to specify the strength of the generated 3D watermark against attacks. The range of `plevel` is: 1, 2, 3, 4 (higher level → more robust 3D watermark).

**Output:**

- `unsigned char ***Vw:` a pointer to a 3D array containing the watermarked volume. The dimensions of the watermarked volume are the same as these of the original one. Attention: this array must be allocated prior to calling the `embedding_sdk()` function.

**Return:**

- `0` : on success. The volume is successfully watermarked.
- `-2001` : as a warning. The volume is successfully watermarked, but due to its small size the watermarked volume has decreased robustness against rotation attacks. One of the dimensions of the original volume is smaller than 128 voxels and thus the algorithm embeds a 3D watermark less robust to rotation attacks.
- `-2002` : as an error. The volume is not watermarked. The `plevel` entered does not belong to the above-mentioned range.
- `-2003` : as an error. The volume is not watermarked. One of the dimensions of the original volume is smaller than 31 voxels and thus the algorithm cannot embed a 3D watermark in it.

**4.3.2 Computing 3D Pick SNR Value.**

The `pick_snr()` function computes the 3D pick SNR value of a grayscale watermarked volume ( $v_w$ ) (or in general an altered volume) with respect to the non-watermarked (original) grayscale volume ( $v$ ).

```
int pick_snr      ( unsigned char ***V, unsigned char ***Vw,
                   int len, int wid, int hei,
                   float *psnr )
```

**Input:**

- `unsigned char ***V:` a pointer to a 3D array containing the non-watermarked (original) volume.
- `unsigned char ***Vw:` a pointer to a 3D array containing the watermarked (altered) volume. Attention: both 3D arrays must have equal dimensions.
- `int len:` the length (in voxels) of both volumes.
- `int wid:` the width (in voxels) of both volumes.
- `int hei:` the height (in voxels) of both volumes.

**Output:**

- `float *psnr:` the 3D pick SNR value in decibels (dB).

**Return:**

- `0` : on success. The 3D pick SNR value is computed successfully.
- `-2004` : as an error. The 3D pick SNR value is not computed. The non-watermarked (original) and the watermarked (altered) volumes are identical and therefore the 3D pick SNR value cannot be computed.

**4.3.3 Detecting 3D Watermarks (basic detection).**

The `basic_detection_sdk()` function performs a basic (fast but not so robust to attacks) search for 3D watermark existence in a grayscale volume (`vw`).

```
int basic_detection_sdk
(
    unsigned char ***Vw,
    int len, int wid, int hei,
    float drthres,
    float *dratio, int *ddecision )
```

**Input:**

- `unsigned char ***Vw:` a pointer to a 3D array containing the volume to be searched for 3D watermark existence.
- `int len:` the length (in voxels) of the volume to be searched.
- `int wid:` the width (in voxels) of the volume to be searched.
- `int hei:` the height (in voxels) of the volume to be searched.
- `float drthres:` the threshold is used to decide whether the volume searched for 3D watermark existence is watermarked with the fixed key or not. The decision is taken by comparing the `drthres` with the `*dratio` (explanation to follow). The range of `drthres` is  $0.5 \rightarrow 1.0$  and the suggested value is 0.720 (experimentally derived).

**Output:**

- `float *dratio:` the detection ratio. The range of `*dratio` is  $0.0 \rightarrow 1.0$ , where 1.0 represents full detection of the 3D watermark under investigation.
- `int *ddecision:` the detection decision. Its value is determined by comparing `drthres` and `*dratio`. If `drthres` is greater than `*dratio`, `*ddecision` returns 0, else `*ddecision` returns 1.

**Return:**

- `0` : on success. The detection procedure terminates successfully.

- -2003 : as an error. The detection procedure terminates unsuccessfully. One of the dimensions of the volume to be searched is smaller than 31 voxels and thus the algorithm cannot search for 3D watermark existence in it.
- -2005 : as an error. The detection procedure terminates unsuccessfully. The `drthres` entered does not belong to the above-mentioned range.

#### 4.3.4 Detecting 3D Watermarks (exhaustive detection).

The `exhaustive_detection_sdk()` function performs an exhaustive, thorough search for 3D watermark existence in a grayscale volume (`Vw`). This function is slower than the previous one (`basic_detection_sdk()`) but enables great robustness against geometric distortions of the given watermarked volume.

```
int exhaustive_detection_sdk
(
    unsigned char ***Vw,
    int len, int wid, int hei,
    float drthres,
    float minsf, float maxsf,
    float *dratio, int *ddecision )
```

##### Input:

- `unsigned char ***Vw`: a pointer to a 3D array containing the volume to be searched for 3D watermark existence
- `int len`: the length (in voxels) of the volume to be searched.
- `int wid`: the width (in voxels) of the volume to be searched.
- `int hei`: the height (in voxels) of the volume to be searched.
- `float drthres`: the threshold is used to decide whether the volume searched for 3D watermark existence is watermarked with the fixed key or not. The decision is taken by comparing the `drthres` with the `*dratio` (explanation to follow). The range of `drthres` is  $0.5 \rightarrow 1.0$  and the suggested value is 0.720 (experimentally derived).
- `float minsf`,
- `float maxsf`: the minimum (`minsf`) and maximum (`maxsf`) scaling factors used to enable search for 3D watermark existence in scaled versions of the volume under investigation. The function searches for 3D watermark existence in successive scaled versions of the volume under investigation bounded by the two scaling factors. Attention: the maximum scaling factor (`maxsf`) must be greater than or equal to the minimum scaling factor (`minsf`).

**Output:**

- float \*dratio: the detection ratio. The range of \*dratio is 0.0 → 1.0, where 1.0 represents full detection of the 3D watermark under investigation.
- int \*ddecision: the detection decision. Its value is determined by comparing drthres and \*dratio. If drthres is greater than \*dratio, \*ddecision returns 0, else \*ddecision returns 1.

**Return:**

- 0 : on success. The detection procedure terminates successfully.
- -2003 : as an error. The detection procedure terminates unsuccessfully. One of the dimensions of the volume to be searched is smaller than 31 voxels and thus the algorithm cannot search for 3D watermark existence in it.
- -2005 : as an error. The detection procedure terminates unsuccessfully. The drthres entered does not belong to the above-mentioned range.

**4.4 Sample Code.**

A sample code is offered below, which demonstrates the usage of all the above-mentioned functions in a simple Windows application.

```
#include <stdio.h>
#include "volmark_sdk.h"

#define v_path      "the name of the path where the volumes are situated"
#define v_filename "the volume filename"
#define vw_filename "the watermarked volume filename"

main
{
    int    len,wid,hei;
    int    plevel;
    float minsf,maxsf;

    // for this example to be functional:
    // - suitable values should be set to all the above-mentioned
    //   variables.
    // - the following functions should be implemented:
    //   - allocate(int,int,int,unsigned char ***) :
    //     a memory allocation function
    //   - free(int,int,int,unsigned char ***) :
    //     a memory deallocation function
    //   - read(char *,char *,int,int,int,unsigned char ***) :
    //     a function reading a volume from a file or a series of files
    //   - save(char *,char *,int,int,int,unsigned char ***) :
    //     a function saving a volume into a file or a series of files
```

```

float psnr;
float drthres;
float dratio;
int ddecision;

unsigned char ***V;
unsigned char ***Vw;

// Embedding 3D Watermarks //////////////////////////////////////

allocate(len, wid, hei, V);
allocate(len, wid, hei, Vw);

read(v_path, v_filename, len, wid, hei, V);

msg = embedding_sdk(V, Vw, len, wid, hei, plevel);
// use msg value to check the termination of the function
if (msg==0) || (msg==-2001) save(v_path, vw_filename, len, wid, hei, Vw);

msg = pick_snr(V, Vw, len, wid, hei, &psnr);
// use msg value to check the termination of the function
if (msg==0) printf("3D Pick SNR = %.3f db", psnr);

free(len, wid, hei, V);
free(len, wid, hei, Vw);

// Detecting 3D Watermarks (basic detection) //////////////////////////////////

allocate(len, wid, hei, Vw);

read(v_path, vw_filename, len, wid, hei, Vw);

msg = basic_detection_sdk(Vw, len, wid, hei, drthres, \
                        &dratio, &ddecision);
// use msg value to check the termination of the function
if (msg==0) printf("Detection Ratio = %.3f |\
                    Detection Decision = %d", dratio, ddecision);

// Detecting 3D Watermarks (exhaustive detection) //////////////////////////////////

msg = exhaustive_detection_sdk(Vw, len, wid, hei, drthres, \
                              minsf, maxsf, &dratio, &ddecision);
// use msg value to check the termination of the function
if (msg==0) printf("Detection Ratio = %.3f |\
                    Detection Decision = %d", dratio, ddecision);

free(len, wid, hei, Vw);
}

```