

Documentation Azure Kinect DK

Azure Kinect DK est un kit de développement doté de capteurs d'intelligence artificielle avancés qui fournissent des modèles vocaux et de vision informatique sophistiqués. Kinect contient un capteur de profondeur, un réseau de microphones spatiaux avec une caméra vidéo et un capteur d'orientation, le tout présenté sous la forme d'un petit appareil tout-en-un proposant plusieurs modes, options et SDK.

À propos d'Azure Kinect DK

VUE D'ENSEMBLE

[Présentation d'Azure Kinect DK](#)

RÉFÉRENCE

[Spécifications matérielles](#)

[Configuration système requise](#)

Bien démarrer

TÉLÉCHARGER

[Bibliothèque NuGet Azure Kinect](#)

[SDK du capteur Azure Kinect](#)

[SDK Azure Kinect Body Tracking](#)

DÉMARRAGE RAPIDE

[Configurer Azure Kinect DK](#)

[Enregistrer des flux de capteur dans un fichier](#)

[Créer votre première application Azure Kinect](#)

[Configurer le SDK Body Tracking](#)

[Créer votre première application Body Tracking](#)

Utiliser le SDK du capteur

GUIDE PRATIQUE

[Rechercher et ouvrir l'appareil](#)

[Récupérer des données d'image](#)

[Accès au microphone](#)

[Enregistrement et lecture](#)

Utiliser le SDK Body Tracking

GUIDE PRATIQUE

[Obtenir les résultats Body Tracking](#)

[Accéder aux données de la structure du corps](#)

Référence

RÉFÉRENCE

[API du SDK Sensor ↗](#)

[API du SDK Body Tracking ↗](#)

[Services de vision ↗](#)

[Services Speech](#)

À propos d'Azure Kinect DK

Article • 31/07/2023



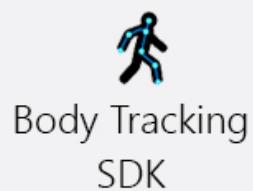
Azure Kinect DK est un SDK doté de capteurs avancés d'intelligence artificielle qui fournissent une vision informatique et des modèles vocaux sophistiqués. Kinect contient un capteur de profondeur, un réseau de microphones spatiaux avec une caméra vidéo et un capteur d'orientation, le tout présenté sous la forme d'un petit appareil tout-en-un doté de plusieurs modes, options et SDK. Il est vendu sur [Microsoft Store en ligne ↗](#).

L'environnement de développement Azure Kinect DK comprend les kits SDK suivants :

- Capteur pour un accès aux appareils et capteurs de bas niveau.
- Suivi des corps pour suivre les corps en 3D.
- Le kit de développement logiciel (SDK) Azure AI Speech pour activer l'accès aux microphones, ainsi que les services vocaux basés sur le cloud Azure.

Par ailleurs, les services Cognitive Vision peuvent être utilisés avec la caméra RVB de l'appareil.

Application



PC



SDK Capteur d'Azure Kinect

Le SDK Capteur d'Azure Kinect donne accès aux capteurs de bas niveau pour la configuration des appareils et des capteurs matériels Azure Kinect DK.

Pour plus d'informations sur le SDK Capteur d'Azure Kinect, consultez [Utilisation du SDK Capteur](#).

Fonctionnalités du SDK Capteur d'Azure Kinect

Les fonctionnalités du SDK Capteur ci-dessous fonctionnent dès lors que le SDK est installé et exécuté sur Azure Kinect DK :

- Accès aux caméras de profondeur et contrôle du mode (mode IR passif et modes de profondeur à champ de vision large et étroit)
- Accès aux caméras RVB et contrôle de celles-ci (par exemple, exposition et balance des blancs)
- Accès aux capteurs de mouvement (gyroscope et accéléromètre)
- Diffusion synchronisée des données des caméras RVB de profondeur avec la possibilité de configurer le délai entre les caméras

- Contrôle de synchronisation des appareils externes avec la possibilité de configurer le décalage entre les appareils
- Accès aux métadonnées de vue de caméra pour la résolution d'image, l'horodatage, etc.
- Accès aux données d'étalonnage des appareils

Outils du SDK Capteur d'Azure Kinect

Le SDK Capteur propose les outils suivants :

- Outil de visualisation permettant de superviser les flux de données des appareils et de configurer les différents modes.
- Un outil d'enregistrement des données de capteur et une API de lecture utilisant le format de conteneur Matroska.
- Un outil de mise à jour de microprogramme Azure Kinect DK.

SDK Suivi des corps d'Azure Kinect

Le SDK Suivi des corps comprend une bibliothèque et un runtime Windows qui permettent de suivre les corps en 3D avec le matériel Azure Kinect DK.

Fonctionnalités du SDK Suivi des corps d'Azure Kinect

Le SDK fourni propose les fonctionnalités de suivi des corps suivantes :

- Assure une segmentation des corps.
- Contient un squelette à l'anatomie correcte pour chaque corps partiel ou complet présent dans le champ de vision.
- Attribue à chaque corps une identité unique.
- Peut assurer le suivi des corps dans le temps.

Outils du SDK Suivi des corps d'Azure Kinect

- Le dispositif de suivi des corps intègre un outil de visualisation destiné à suivre les corps en 3D.

Kit de développement logiciel (SDK) Azure AI Speech

Le SDK Speech permet d'activer les services vocaux connectés à Azure.

Speech Services

- Reconnaissance vocale
- Traduction vocale
- Synthèse vocale

ⓘ Notes

Azure Kinect DK n'est pas équipé de haut-parleurs.

Pour plus d'informations, consultez la [documentation Speech Service](#).

Services de vision

[Azure Cognitive Vision Services](#) propose des services Azure capables d'identifier et d'analyser le contenu des images et des vidéos.

- [Vision par ordinateur](#)
- [Visage](#)
- [Video Indexer](#)
- [Content Moderator](#)
- [Custom Vision](#)

Étant donné que les services évoluent et s'améliorent en permanence, vérifiez régulièrement s'il n'existe pas de nouveaux services [Azure AI services](#) pour améliorer votre application. Pour obtenir des informations à jour sur les derniers services disponibles, consultez la page des [labos Azure AI services](#).

Configuration matérielle requise pour Azure Kinect

Azure Kinect DK intègre les dernières technologies de capteur de Microsoft dans un même accessoire connecté par USB. Pour plus d'informations, consultez [Spécifications matérielles Azure Kinect DK](#).

Étapes suivantes

Maintenant que vous avez une idée globale d'Azure Kinect DK, le moment est venu d'aller plus loin et de le configurer.

Démarrage rapide : Configurer Azure Kinect DK

Démarrage rapide : Configurer Azure Kinect DK

Article • 01/06/2023

Ce guide de démarrage rapide fournit des instructions sur la configuration d'Azure Kinect DK. Nous allons vous montrer comment tester la visualisation du flux de capteur et utiliser la [visionneuse Kinect Azure](#).

Si vous n'avez pas d'abonnement Azure, créez un [compte gratuit](#) avant de commencer.

Configuration requise

Consultez la [configuration requise](#) pour voir si la configuration de votre ordinateur hôte répond aux exigences minimales d'Azure Kinect DK.

Configurer le matériel

ⓘ Notes

Veillez à supprimer le film de protection de la caméra avant d'utiliser l'appareil.

1. Branchez le câble d'alimentation sur la prise jack située à l'arrière de l'appareil. Connectez l'adaptateur secteur USB à l'autre extrémité du câble, puis branchez l'adaptateur à une prise d'alimentation.
2. Connectez le câble de données USB à votre appareil, puis à un port USB 3.0 de votre PC.

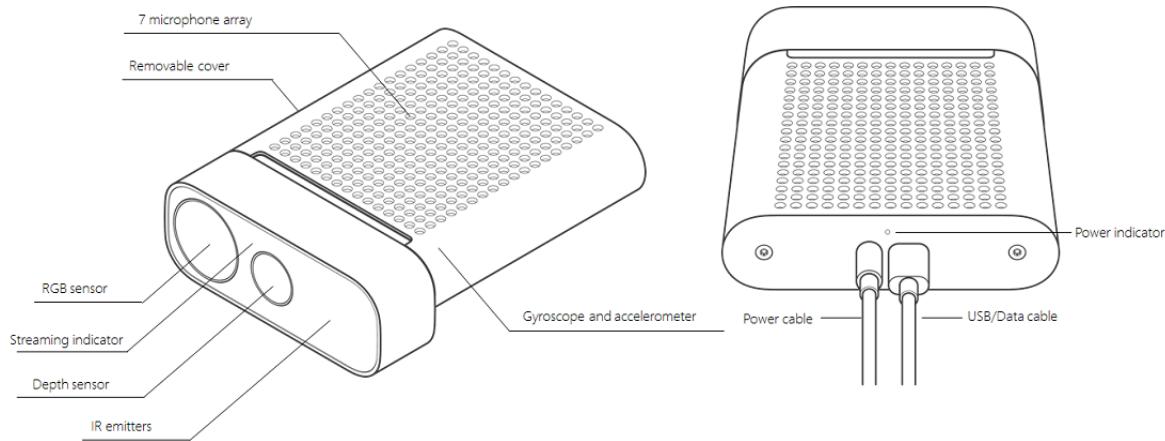
ⓘ Notes

Pour de meilleurs résultats, connectez le câble directement à l'appareil et au PC. Évitez d'utiliser des extensions ou des adaptateurs supplémentaires dans la connexion.

3. Vérifiez que le voyant d'alimentation (situé à côté du câble USB) est blanc et fixe.

La mise sous tension de l'appareil prend quelques secondes. L'appareil est prêt à être utilisé lorsque le voyant de diffusion des données situé à l'avant s'éteint.

Pour plus d'informations sur le voyant d'alimentation, consultez [À quoi correspond ce voyant ?](#)



Télécharger le Kit de développement logiciel (SDK)

1. Sélectionnez le lien [Télécharger le SDK](#).
2. Installez le SDK sur votre PC.

Mettre à jour le microprogramme

Pour fonctionner correctement, le SDK a besoin de la dernière version du microprogramme de l'appareil. Pour vérifier et mettre à jour la version de votre microprogramme, suivez les étapes décrites dans [Mettre à jour le microprogramme Azure Kinect DK](#).

Vérifier que l'appareil diffuse des données

1. Lancez la [visionneuse Azure Kinect](#). Vous pouvez démarrer cet outil à l'aide de l'une des méthodes suivantes :
 - Vous pouvez lancer la visionneuse à partir de la ligne de commande ou en double-cliquant sur le fichier exécutable. Le fichier `k4aviewer.exe` se trouve dans le répertoire des outils du SDK (par exemple, `C:\Program Files\Azure Kinect SDK vX.Y.Z\tools\k4aviewer.exe`, où `X.Y.Z` correspond à la version installée du SDK).

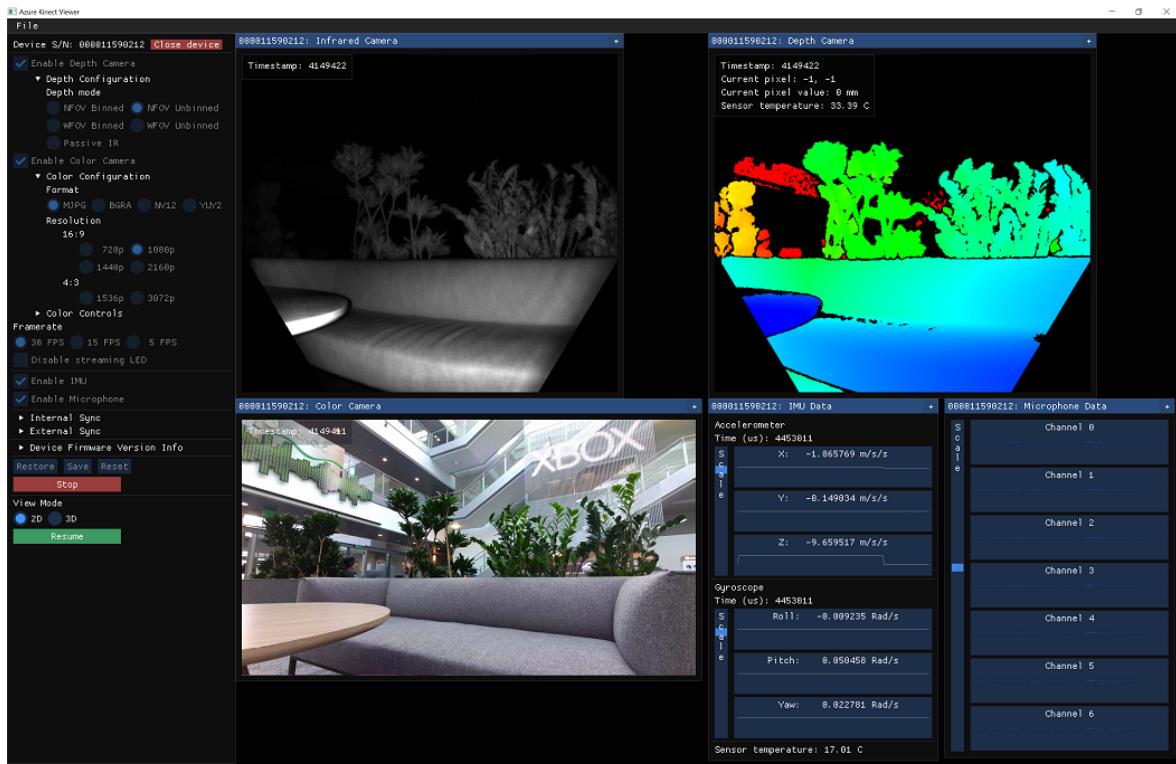
- Vous pouvez lancer la visionneuse Azure Kinect à partir du menu Démarrer de l'appareil.

2. Dans la visionneuse Azure Kinect, sélectionnez Open Device (Ouvrir l'appareil)>Start (Démarrer) .



3. Vérifiez que l'outil visualise chaque flux de capteur :

- Caméra à profondeur de champ
- Caméra couleur
- Caméra infrarouge
- Unités de mesure inertielle
- Microphones



Vous avez terminé la configuration d’Azure Kinect DK. Vous pouvez maintenant commencer à développer votre application ou à intégrer des services.

Si vous rencontrez des problèmes, consultez [Résolution des problèmes](#).

Voir aussi

- Informations sur le matériel Azure Kinect DK
- Mettre à jour le microprogramme de l’appareil
- En savoir plus sur la [visionneuse Azure Kinect](#)

Étapes suivantes

Une fois que le service Azure Kinect DK est prêt et opérationnel, vous pouvez également apprendre à :

[Enregistrer des flux de capteur dans un fichier](#)

Démarrage rapide : Enregistrer les flux de capteur Azure Kinect dans un fichier

Article • 01/06/2023

Ce guide de démarrage rapide fournit des informations sur la façon dont vous pouvez utiliser l'outil [enregistreur Azure Kinect](#) pour enregistrer les flux de données du SDK Capteur dans un fichier.

Si vous n'avez pas d'abonnement Azure, créez un [compte gratuit](#) avant de commencer.

Prérequis

Ce guide de démarrage rapide suppose que :

- Azure Kinect DK est connecté à votre PC hôte et correctement alimenté.
- Vous avez terminé la [configuration](#) du matériel.

Créer un enregistrement

1. Ouvrez une invite de commandes et indiquez le chemin de l'[enregistreur Azure Kinect](#), qui figure dans le répertoire des outils installés sous le nom `k4arecorder.exe`. Par exemple : `C:\Program Files\Azure Kinect SDK\tools\k4arecorder.exe`.

2. Faites un enregistrement d'une durée de 5 secondes.

```
k4arecorder.exe -l 5 %TEMP%\output.mkv
```

3. Par défaut, l'enregistreur utilise le mode de profondeur NFOV sans compartimentation et génère 1 080 p RVB à 30 i/s avec des données IMU. Pour obtenir une vue d'ensemble complète des options d'enregistrement ainsi que des conseils, consultez [Enregistreur Azure Kinect](#).

Lire l'enregistrement

Vous pouvez utiliser la [visionneuse Azure Kinect](#) pour lire un enregistrement.

1. Lancez `k4aviewer.exe`.

2. Déroulez l'onglet **Open Recording** (Ouvrir un enregistrement) et ouvrez votre enregistrement.

Étapes suivantes

Maintenant que vous savez comment enregistrer les flux de capteur dans un fichier, il est temps de

[Créer votre première application Azure Kinect](#)

Démarrage rapide : Créer votre première application Azure Kinect

Article • 01/06/2023

Vous débutez avec Azure Kinect DK ? Ce guide de démarrage rapide va vous aider à devenir opérationnel avec l'appareil.

Si vous n'avez pas d'abonnement Azure, créez un [compte gratuit](#) avant de commencer.

Voici les fonctions qui sont abordées ici :

- [k4a_device_get_installed_count\(\)](#)
- [k4a_device_open\(\)](#)
- [k4a_device_get_serialnum\(\)](#)
- [k4a_device_start_cameras\(\)](#)
- [k4a_device_stop_cameras\(\)](#)
- [k4a_device_close\(\)](#)

Prérequis

1. [Configurer l'appareil Azure Kinect DK.](#)
2. [Télécharger](#) et installer le SDK Capteur d'Azure Kinect.

En-têtes

Le seul en-tête dont vous avez besoin est `k4a.h`. Veillez à ce que le compilateur que vous avez choisi est configuré avec les dossiers lib et include du SDK. Vous devez aussi lier les fichiers `k4a.lib` et `k4a.dll`. Consultez éventuellement [Ajouter la bibliothèque Azure Kinect à votre projet.](#)

```
C
```

```
#include <k4a/k4a.h>
```

Rechercher un appareil Azure Kinect DK

Plusieurs appareils Azure Kinect DK peuvent être connectés à votre ordinateur. Nous allons commencer par déterminer leur nombre, si tant est qu'il y en ait, à l'aide de la

fonction [k4a_device_get_installed_count\(\)](#). Cette fonction doit en principe fonctionner directement, sans étapes de configuration supplémentaires.

C

```
uint32_t count = k4a_device_get_installed_count();
```

Dès lors que vous avez déterminé qu'un appareil est connecté à l'ordinateur, vous pouvez l'ouvrir avec [k4a_device_open\(\)](#). Vous pouvez fournir l'index de l'appareil à ouvrir ou utiliser simplement `K4A_DEVICE_DEFAULT` pour le premier.

C

```
// Open the first plugged in Kinect device
k4a_device_t device = NULL;
k4a_device_open(K4A_DEVICE_DEFAULT, &device);
```

Comme c'est généralement le cas avec la bibliothèque Azure Kinect, quand vous ouvrez un de ses composants, vous devez aussi le refermer après l'avoir utilisé. Pour cela, pensez à appeler [k4a_device_close\(\)](#).

C

```
k4a_device_close(device);
```

Une fois l'appareil ouvert, vous pouvez le tester pour vérifier qu'il fonctionne. C'est le cas si vous pouvez lire le numéro de série de l'appareil.

C

```
// Get the size of the serial number
size_t serial_size = 0;
k4a_device_get_serialnum(device, NULL, &serial_size);

// Allocate memory for the serial, then acquire it
char *serial = (char*)(malloc(serial_size));
k4a_device_get_serialnum(device, serial, &serial_size);
printf("Opened device: %s\n", serial);
free(serial);
```

Démarrer les caméras

Une fois que vous avez ouvert l'appareil, vous devez configurer la caméra avec un objet [k4a_device_configuration_t](#). Il existe différentes options de configuration de la caméra.

Choisissez les paramètres qui correspondent le mieux à votre propre scénario.

```
C

// Configure a stream of 4096x3072 BRGA color data at 15 frames per second
k4a_device_configuration_t config = K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;
config.camera_fps      = K4A_FRAMES_PER_SECOND_15;
config.color_format    = K4A_IMAGE_FORMAT_COLOR_BGRA32;
config.color_resolution = K4A_COLOR_RESOLUTION_3072P;

// Start the camera with the given configuration
k4a_device_start_cameras(device, &config);

// ...Camera capture and application specific code would go here...

// Shut down the camera when finished with application logic
k4a_device_stop_cameras(device);
```

Gestion des erreurs

Dans un souci de clarté et de concision, la gestion des erreurs n'apparaît pas dans les exemples fournis. Cependant, la gestion des erreurs est toujours importante ! De nombreuses fonctions peuvent retourner un type de réussite/échec général [k4a_result_t](#) ou une variante plus spécifiques avec des informations détaillées comme [k4a_wait_result_t](#). Consultez la documentation ou IntelliSense pour chaque fonction de façon à vous faire une idée des messages d'erreur que vous êtes susceptible de rencontrer.

Vous pouvez utiliser les macros [K4A_SUCCEEDED](#) et [K4A_FAILED](#) pour vérifier le résultat d'une fonction. Au lieu d'ouvrir simplement un appareil Azure Kinect DK, vous pouvez protéger l'appel de fonction comme ceci :

```
C

// Open the first plugged in Kinect device
k4a_device_t device = NULL;
if ( K4A_FAILED( k4a_device_open(K4A_DEVICE_DEFAULT, &device) ) )
{
    printf("Failed to open k4a device!\n");
    return;
}
```

Source complète

```
C
```

```

#pragma comment(lib, "k4a.lib")
#include <k4a/k4a.h>

#include <stdio.h>
#include <stdlib.h>

int main()
{
    uint32_t count = k4a_device_get_installed_count();
    if (count == 0)
    {
        printf("No k4a devices attached!\n");
        return 1;
    }

    // Open the first plugged in Kinect device
    k4a_device_t device = NULL;
    if (K4A_FAILED(k4a_device_open(K4A_DEVICE_DEFAULT, &device)))
    {
        printf("Failed to open k4a device!\n");
        return 1;
    }

    // Get the size of the serial number
    size_t serial_size = 0;
    k4a_device_get_serialnum(device, NULL, &serial_size);

    // Allocate memory for the serial, then acquire it
    char *serial = (char*)(malloc(serial_size));
    k4a_device_get_serialnum(device, serial, &serial_size);
    printf("Opened device: %s\n", serial);
    free(serial);

    // Configure a stream of 4096x3072 BRGA color data at 15 frames per
    second
    k4a_device_configuration_t config = K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;
    config.camera_fps      = K4A_FRAMES_PER_SECOND_15;
    config.color_format    = K4A_IMAGE_FORMAT_COLOR_BGRA32;
    config.color_resolution = K4A_COLOR_RESOLUTION_3072P;

    // Start the camera with the given configuration
    if (K4A_FAILED(k4a_device_start_cameras(device, &config)))
    {
        printf("Failed to start cameras!\n");
        k4a_device_close(device);
        return 1;
    }

    // Camera capture and application specific code would go here

    // Shut down the camera when finished with application logic
    k4a_device_stop_cameras(device);
    k4a_device_close(device);

```

```
    return 0;  
}
```

Étapes suivantes

Découvrir comment rechercher et ouvrir un appareil Azure Kinect DK à l'aide du SDK Capteur

[Rechercher et ouvrir un appareil](#)

Démarrage rapide : Configurer le suivi des corps Azure Kinect

Article • 01/06/2023

Ce guide de démarrage rapide vous accompagne tout au long du processus de mise en service du suivi des corps sur votre appareil Azure Kinect DK.

Configuration système requise

Le PC hôte du SDK Suivi des corps doit être équipé d'un GPU NVIDIA. La configuration recommandée du PC hôte du SDK Suivi des corps est décrite dans la page [Configuration système requise](#).

Installer les logiciels

[Installer la dernière version du pilote NVIDIA](#) ↗

Téléchargez et installez la dernière version du pilote NVIDIA de votre carte graphique. Les pilotes plus anciens risquent de ne pas être compatibles avec les binaires CUDA redistribués avec le SDK Suivi des corps.

[Package redistribuable Visual C++ pour Visual Studio 2015](#) ↗

Téléchargez et installez Redistributable Visual C++ pour Visual Studio 2015.

Configurer le matériel

[Configurer Azure Kinect DK](#)

Lancez la [visionneuse Azure Kinect](#) pour vérifier que votre appareil Azure Kinect DK est correctement configuré.

Télécharger le SDK Suivi des corps

1. Sélectionnez le lien de [téléchargement du SDK Suivi des corps](#).

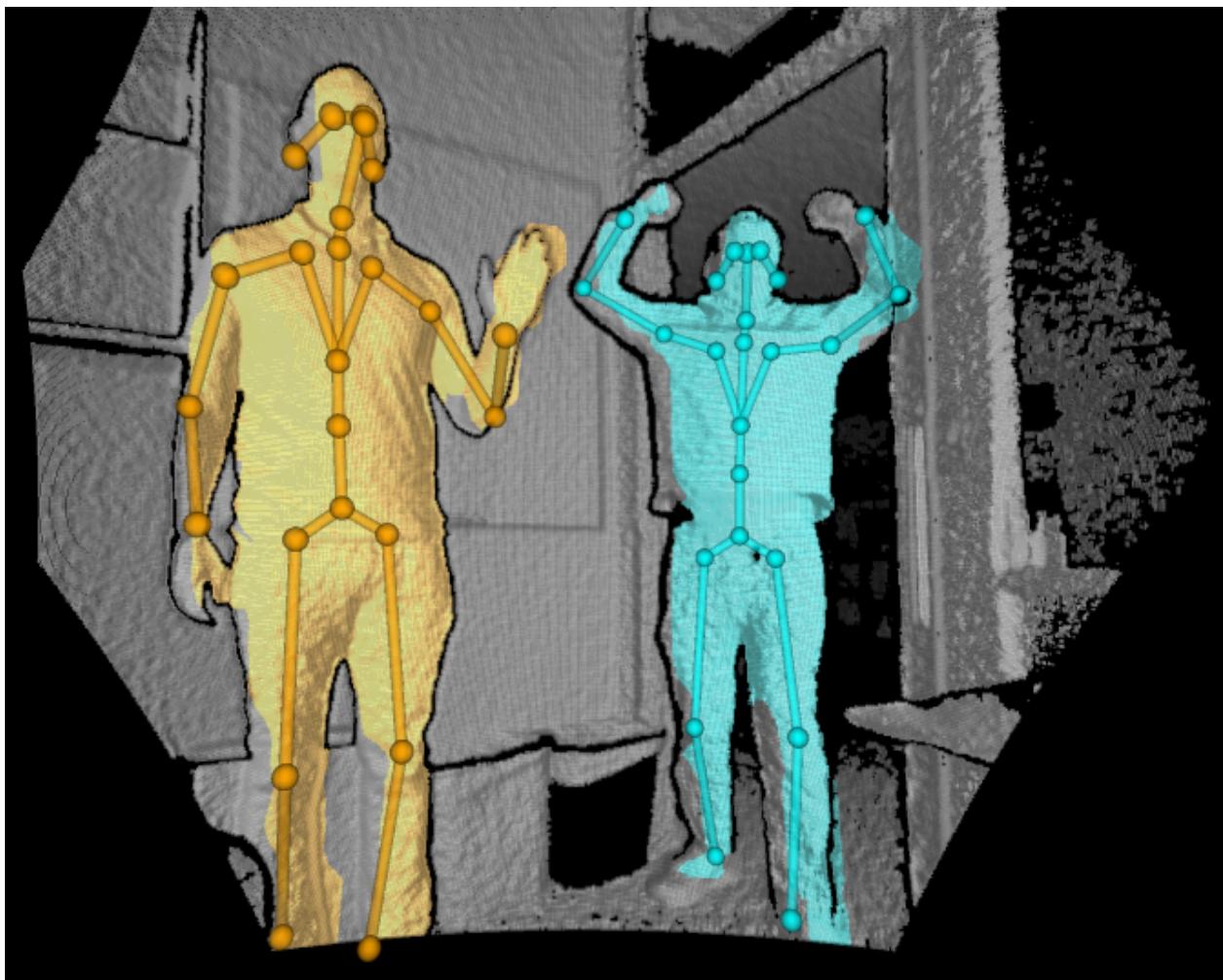
2. Installez le SDK Suivi des corps sur votre PC.

Vérifier le suivi des corps

Lancez la **visionneuse de suivi des corps Azure Kinect** pour vérifier que le SDK Suivi des corps est correctement configuré. La visionneuse s'installe avec le programme d'installation msi du SDK. Elle se trouve dans le menu de démarrage ou dans `<SDK Installation Path>\tools\k4abt_simple_3d_viewer.exe`.

Si votre GPU n'est pas suffisamment puissant et que vous souhaitez quand même tester le résultat, vous pouvez lancer la **visionneuse de suivi des corps Azure Kinect** sur la ligne de commande en utilisant la commande suivante : `<SDK Installation Path>\tools\k4abt_simple_3d_viewer.exe CPU`

Si tout est correctement configuré, une fenêtre contenant un nuage de points 3D et les corps suivis s'affiche à l'écran.



Spécification de l'environnement d'exécution du runtime ONNX

Le kit de développement logiciel (SDK) de suivi du corps prend en charge les environnements d'exécution du processeur, CUDA, DirectML (Windows uniquement) et TensorRT pour inférer le modèle d'estimation de pose. La valeur `K4ABT_TRACKER_PROCESSING_MODE_GPU` passe par défaut à l'exécution de CUDA sur Linux et à l'exécution de DirectML sur Windows. Trois modes supplémentaires ont été ajoutés pour sélectionner des environnements d'exécution spécifiques : `K4ABT_TRACKER_PROCESSING_MODE_GPU_CUDA`, `K4ABT_TRACKER_PROCESSING_MODE_GPU_DIRECTML` et `K4ABT_TRACKER_PROCESSING_MODE_GPU_TENSORRT`.

ⓘ Notes

Le runtime ONNX affiche des avertissements pour les opcodes qui ne sont pas accélérés. Vous pouvez l'ignorer en toute sécurité.

Le runtime ONNX contient des variables d'environnement pour contrôler la mise en cache du modèle TensorRT. Les valeurs recommandées sont les suivantes :

- `ORT_TENSORRT_ENGINE_CACHE_ENABLE=1`
- `ORT_TENSORRT_CACHE_PATH="pathname"`

Le dossier doit être créé avant de commencer le suivi du corps.

ⓘ Important

TensorRT pré-traite le modèle avant l'inférence entraînant des temps de démarrage étendus par rapport à d'autres environnements d'exécution. Le service de mise en cache du moteur limite ce nombre à la première exécution, mais il est expérimental et est spécifique au modèle, à la version du runtime ONNX, à la version TensorRT et au modèle GPU.

L'environnement d'exécution TensorRT prend en charge FP32 (par défaut) et FP16. Le niveau de performance de FP16 est 2 fois plus important pour une diminution minimale de la précision. Pour spécifier FP16 :

- `ORT_TENSORRT_FP16_ENABLE=1`

Bibliothèques de liens dynamiques requises pour les environnements d'exécution du runtime ONNX

| Mode | ORT 1.10 | CUDA 11.4.3 | CUDNN 8.2.2.26 | TensorRT 8.0.3.4 |
|-------------|--------------------------------|--------------------------|-----------------------|-------------------------|
| UC | msvcp140 | - | - | - |
| | onnxruntime | | | |
| CUDA | msvcp140 | cudart64_110 | cudnn64_8 | - |
| | onnxruntime | cufft64_10 | cudnn_ops_infer64_8 | |
| | onnxruntime_providers_cuda | cublas64_11 | cudnn_cnn_infer64_8 | |
| | onnxruntime_providers_shared | cublasLt64_11 | | |
| DirectML | msvcp140 | - | - | - |
| | onnxruntime | | | |
| | directml | | | |
| TensorRT | msvcp140 | cudart64_110 | - | nvinfer |
| | onnxruntime | cufft64_10 | | nvinfer_plugin |
| | onnxruntime_providers_cuda | cublas64_11 | | |
| | onnxruntime_providers_shared | cublasLt64_11 | | |
| | onnxruntime_providers_tensorrt | nvrtc64_112_0 | | |
| | | nvrtc- builtins64_114 | | |

Exemples

Les exemples d'utilisation du SDK Suivi des corps se trouvent [ici ↗](#).

Étapes suivantes

[Créer votre première application de suivi des corps](#)

Démarrage rapide : Créer une application de suivi des corps Azure Kinect

Article • 01/06/2023

Vous débutez avec le SDK Suivi des corps ? Ce guide de démarrage rapide va vous aider à devenir opérationnel avec le suivi des corps. Vous trouverez d'autres exemples dans le dépôt [Azure-Kinect-Sample](#).

Prérequis

- Configurer Azure Kinect DK
- Configurer le SDK Suivi des corps
- Découvrez pas à pas comment [créer votre première application Azure Kinect](#).
- Familiarisez-vous avec les fonctions suivantes du SDK Capteur :
 - [k4a_device_open\(\)](#)
 - [k4a_device_start_cameras\(\)](#)
 - [k4a_device_stop_cameras\(\)](#)
 - [k4a_device_close\(\)](#)
- Consultez la documentation sur les fonctions suivantes du SDK Suivi des corps :
 - [k4abt_tracker_create\(\)](#)
 - [k4abt_tracker_enqueue_capture\(\)](#)
 - [k4abt_tracker_pop_result\(\)](#)
 - [k4abt_tracker_shutdown\(\)](#)
 - [k4abt_tracker_destroy\(\)](#)

En-têtes

Le suivi des corps utilise un seul en-tête : `k4abt.h`. Incluez cet en-tête en plus de `k4a.h`. Veillez à ce que le compilateur que vous avez choisi est configuré pour les dossiers `lib` et `include` des SDK Capteur et Suivi des corps. Vous devez aussi établir un lien avec les fichiers `k4a.lib` et `k4abt.lib`. L'exécution de l'application exige la présence de `k4a.dll`, `k4abt.dll`, `onnxruntime.dll` et `dnn_model.onnx` dans le chemin d'exécution de l'application.

```
#include <k4a/k4a.h>
#include <k4abt.h>
```

Ouvrir l'appareil et démarrer la caméra

Pour votre première application de suivi des corps, un seul appareil Azure Kinect est censé être connecté au PC.

Le suivi des corps repose sur le SDK Capteur. Pour utiliser le suivi des corps, vous devez d'abord ouvrir et configurer l'appareil. Utilisez la fonction [k4a_device_open\(\)](#) pour ouvrir l'appareil et le configurer ensuite avec un objet [k4a_device_configuration_t](#). Pour de meilleurs résultats, définissez le mode de profondeur sur

`K4A_DEPTH_MODE_NFOV_UNBINNED` ou `K4A_DEPTH_MODE_WFOV_2X2BINNED`. L'outil de suivi des corps ne s'exécute pas si le mode de profondeur est défini sur `K4A_DEPTH_MODE_OFF` ou `K4A_DEPTH_MODE_PASSIVE_IR`.

Vous trouverez des informations complémentaires sur la recherche et l'ouverture de l'appareil dans [cette page](#).

Les modes de profondeur d'Azure Kinect sont décrits de façon plus détaillée dans les pages suivantes : [spécification matérielle](#) et énumérations [k4a_depth_mode_t](#).

C

```
k4a_device_t device = NULL;
k4a_device_open(0, &device);

// Start camera. Make sure depth camera is enabled.
k4a_device_configuration_t deviceConfig =
K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;
deviceConfig.depth_mode = K4A_DEPTH_MODE_NFOV_UNBINNED;
deviceConfig.color_resolution = K4A_COLOR_RESOLUTION_OFF;
k4a_device_start_cameras(device, &deviceConfig);
```

Créer le dispositif de suivi

Pour obtenir les résultats du suivi des corps, vous devez dans un premier temps créer un dispositif de suivi des corps. Pour cela, vous avez besoin de la structure d'étalonnage de capteur [k4a_calibration_t](#). L'étalonnage de capteur peut être interrogé à l'aide de la fonction [k4a_device_get_calibration\(\)](#).

C

```
k4a_calibration_t sensor_calibration;
k4a_device_get_calibration(device, deviceConfig.depth_mode,
deviceConfig.color_resolution, &sensor_calibration);

k4abt_tracker_t tracker = NULL;
k4abt_tracker_configuration_t tracker_config = K4ABT_TRACKER_CONFIG_DEFAULT;
k4abt_tracker_create(&sensor_calibration, tracker_config, &tracker);
```

Obtenir les captures de l'appareil Azure Kinect

Vous trouverez des informations complémentaires sur la récupération de données d'image dans [cette page](#).

C

```
// Capture a depth frame
k4a_capture_t sensor_capture;
k4a_device_get_capture(device, &sensor_capture, TIMEOUT_IN_MS);
```

Empiler la capture et dépiler les résultats

Le dispositif de suivi gère en interne une file d'attente d'entrée et une file d'attente de sortie pour traiter de manière asynchrone et plus efficace les captures d'Azure Kinect DK. L'étape suivante consiste à utiliser la fonction `k4abt_tracker_enqueue_capture()` pour ajouter une nouvelle capture à la file d'attente d'entrée. Utilisez la fonction `k4abt_tracker_pop_result()` pour dépiler un résultat de la file d'attente de sortie. La valeur de délai d'attente dépend de l'application et contrôle le temps d'attente de mise en file d'attente.

Votre première application de suivi des corps utilise le modèle de traitement en temps réel. Pour obtenir une description détaillée des autres modèles, consultez [Obtenir les résultats du suivi des corps](#).

C

```
k4a_wait_result_t queue_capture_result =
k4abt_tracker_enqueue_capture(tracker, sensor_capture, K4A_WAIT_INFINITE);
k4a_capture_release(sensor_capture); // Remember to release the sensor
capture once you finish using it
if (queue_capture_result == K4A_WAIT_RESULT_FAILED)
{
    printf("Error! Adding capture to tracker process queue failed!\n");
    break;
}
```

```
k4abt_frame_t body_frame = NULL;
k4a_wait_result_t pop_frame_result = k4abt_tracker_pop_result(tracker,
&body_frame, K4A_WAIT_INFINITE);
if (pop_frame_result == K4A_WAIT_RESULT_SUCCEEDED)
{
    // Successfully popped the body tracking result. Start your processing
    ...
    k4abt_frame_release(body_frame); // Remember to release the body frame
    once you finish using it
}
```

Accéder aux données de résultat du suivi des corps

Les résultats du suivi des corps pour chaque capture de capteur sont stockés dans une structure de trame corporelle [k4abt_frame_t](#). Chaque trame corporelle est constituée de trois composantes clés : une collection de structures corporelles, un mappage d'index de corps 2D et la capture d'entrée.

Votre première application de suivi des corps accède uniquement au nombre de corps détectés. Pour obtenir une description détaillée des données contenues dans une trame corporelle, consultez [Accéder aux données d'une trame corporelle](#).

C

```
size_t num_bodies = k4abt_frame_get_num_bodies(body_frame);
printf("%zu bodies are detected!\n", num_bodies);
```

Nettoyer

La dernière étape consiste à arrêter le dispositif de suivi des corps et à libérer l'objet de suivi des corps. Vous devez aussi arrêter et fermer l'appareil.

C

```
k4abt_tracker_shutdown(tracker);
k4abt_tracker_destroy(tracker);
k4a_device_stop_cameras(device);
k4a_device_close(device);
```

Source complète

C

```
#include <stdio.h>
#include <stdlib.h>

#include <k4a/k4a.h>
#include <k4abt.h>

#define VERIFY(result, error)
\
    if(result != K4A_RESULT_SUCCEEDED)
\
    {
\
        printf("%s \n - (File: %s, Function: %s, Line: %d)\n", error,
__FILE__, __FUNCTION__, __LINE__); \
        exit(1);
\
    }
\
}

int main()
{
    k4a_device_t device = NULL;
VERIFY(k4a_device_open(0, &device), "Open K4A Device failed!");

    // Start camera. Make sure depth camera is enabled.
    k4a_device_configuration_t deviceConfig =
K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;
    deviceConfig.depth_mode = K4A_DEPTH_MODE_NFOV_UNBINNED;
    deviceConfig.color_resolution = K4A_COLOR_RESOLUTION_OFF;
    VERIFY(k4a_device_start_cameras(device, &deviceConfig), "Start K4A
cameras failed!");

    k4a_calibration_t sensor_calibration;
    VERIFY(k4a_device_get_calibration(device, deviceConfig.depth_mode,
deviceConfig.color_resolution, &sensor_calibration),
"Get depth camera calibration failed!");

    k4abt_tracker_t tracker = NULL;
    k4abt_tracker_configuration_t tracker_config =
K4ABT_TRACKER_CONFIG_DEFAULT;
    VERIFY(k4abt_tracker_create(&sensor_calibration, tracker_config,
&tracker), "Body tracker initialization failed!");

    int frame_count = 0;
    do
    {
        k4a_capture_t sensor_capture;
        k4a_wait_result_t get_capture_result =
k4a_device_get_capture(device, &sensor_capture, K4A_WAIT_INFINITE);
        if (get_capture_result == K4A_WAIT_RESULT_SUCCEEDED)
        {
            frame_count++;
        }
    }
}
```

```

        k4a_wait_result_t queue_capture_result =
k4abt_tracker_enqueue_capture(tracker, sensor_capture, K4A_WAIT_INFINITE);
        k4a_capture_release(sensor_capture); // Remember to release the
sensor capture once you finish using it
        if (queue_capture_result == K4A_WAIT_RESULT_TIMEOUT)
{
        // It should never hit timeout when K4A_WAIT_INFINITE is
set.
        printf("Error! Add capture to tracker process queue
timeout!\n");
        break;
}
else if (queue_capture_result == K4A_WAIT_RESULT_FAILED)
{
        printf("Error! Add capture to tracker process queue
failed!\n");
        break;
}

        k4abt_frame_t body_frame = NULL;
        k4a_wait_result_t pop_frame_result =
k4abt_tracker_pop_result(tracker, &body_frame, K4A_WAIT_INFINITE);
        if (pop_frame_result == K4A_WAIT_RESULT_SUCCEEDED)
{
        // Successfully popped the body tracking result. Start your
processing

        size_t num_bodies = k4abt_frame_get_num_bodies(body_frame);
printf("%zu bodies are detected!\n", num_bodies);

        k4abt_frame_release(body_frame); // Remember to release the
body frame once you finish using it
}
else if (pop_frame_result == K4A_WAIT_RESULT_TIMEOUT)
{
        // It should never hit timeout when K4A_WAIT_INFINITE is
set.
        printf("Error! Pop body frame result timeout!\n");
        break;
}
else
{
        printf("Pop body frame result failed!\n");
        break;
}
}

else if (get_capture_result == K4A_WAIT_RESULT_TIMEOUT)
{
        // It should never hit time out when K4A_WAIT_INFINITE is set.
printf("Error! Get depth frame time out!\n");
        break;
}
else
{
        printf("Get depth capture returned error: %d\n",

```

```
get_capture_result);
    break;
}

} while (frame_count < 100);

printf("Finished body tracking processing!\n");

k4abt_tracker_shutdown(tracker);
k4abt_tracker_destroy(tracker);
k4a_device_stop_cameras(device);
k4a_device_close(device);

return 0;
}
```

Étapes suivantes

Obtenir les résultats du suivi des corps

Caméra de profondeur d'Azure Kinect DK

Article • 01/06/2023

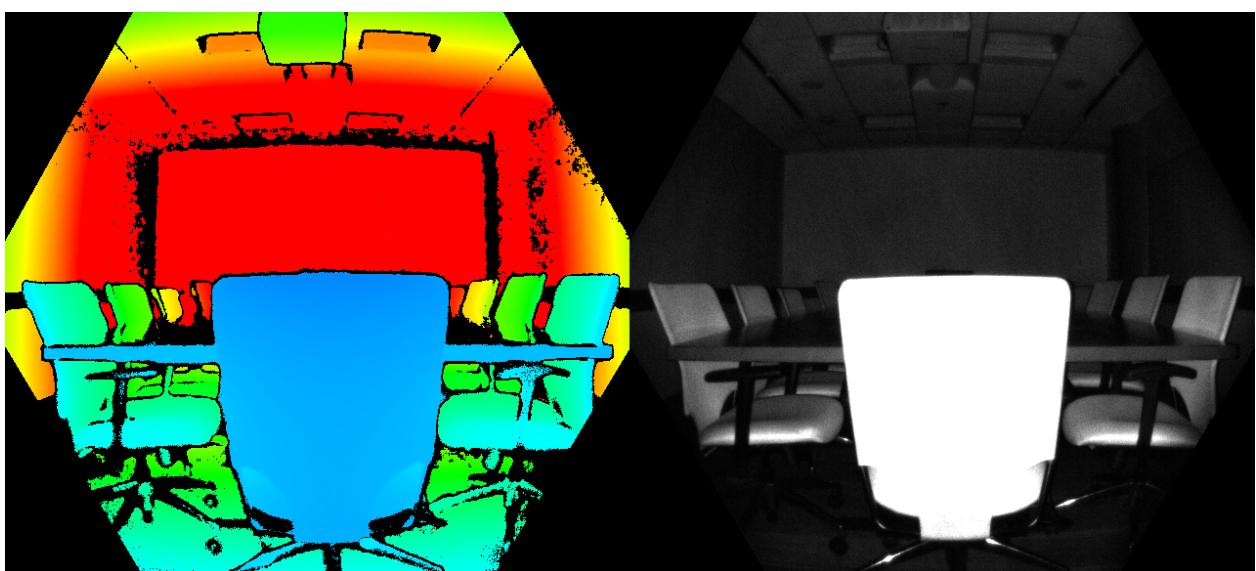
Cette page explique comment utiliser la caméra de profondeur de votre Azure Kinect DK. La caméra de profondeur est la seconde des deux caméras. Comme décrit dans les sections précédentes, l'autre caméra est la caméra RVB (couleur).

Principes de fonctionnement

La caméra de profondeur d'Azure Kinect DK implémente le principe de temps de vol (ToF, Time-of-Flight) d'AMCW (Amplitude Modulated Continuous Wave). Elle convertit l'illumination modulée de la scène dans le spectre proche de l'IR (NIR). Elle enregistre ensuite une mesure indirecte du temps que la lumière prend pour parcourir le trajet aller-retour de la caméra à la scène.

Ces mesures sont traitées pour générer une carte de profondeur. Une carte de profondeur est un ensemble de valeurs de coordonnée Z pour chaque pixel de l'image, mesurée en millimètres.

À côté de la carte de profondeur, nous obtenons également une lecture IR dite propre. La valeur des pixels dans la lecture IR propre est proportionnelle à la quantité de lumière renvoyée par la scène. L'image ressemble à une image IR normale. La figure ci-dessous montre un exemple de carte de profondeur (à gauche) et l'image IR propre correspondante (à droite).



Fonctionnalités clés

Les caractéristiques techniques de la caméra de profondeur sont les suivantes :

- Un processeur d'image ToF de 1 mégapixel doté d'une technologie de pixel avancée offrant des fréquences de modulation et une précision de profondeur plus élevées
- Deux diodes laser NIR permettant d'obtenir des modes de profondeur de champ de vision (FoV) proches et larges.
- Le plus petit pixel ToF au monde, de 3,5 µm sur 3,5 µm.
- Une sélection automatique de gain par pixel activant une vaste plage dynamique permettant une capture propre d'objets proches et distants.
- Un obturateur global permettant d'améliorer les performances à la lumière du jour.
- Une méthode de calcul de profondeur en plusieurs phases qui permet d'obtenir une précision stable, même en présence de variations de processeur, de laser et d'alimentation.
- Un faible taux d'erreurs systématiques et aléatoires.



La caméra de profondeur transmet des images IR modulées brutes au PC hôte. Sur le PC, le logiciel du moteur de profondeur accéléré par GPU convertit le signal brut en cartes de profondeur. La caméra de profondeur prend en charge plusieurs modes. Les modes **Champ de vision (FoV) étroit** sont idéaux pour les scènes faiblement étendues dans les dimensions X et Y, mais plus étendues dans la dimension Z. Si la scène comprend des étendues X et Y importantes, mais des plages Z plus petites, les modes **FoV large** sont plus adaptés.

La caméra de profondeur prend en charge les **modes de compartimentage 2 x 2** pour étendre la plage Z par rapport aux **modes sans compartimentage** correspondants. Le compartimentage s'effectue au détriment de la résolution de l'image. Tous les modes

peuvent être exécutés jusqu'à une fréquence de 30 images par seconde (i/s), à l'exception du mode 1 mégapixel (MP) qui s'exécute à une fréquence d'images maximale de 15 i/s. La caméra de profondeur offre également un **mode IR passif**. Dans ce mode, les illuminateurs de la caméra ne sont pas actifs et seul l'éclairage ambiant est pris en compte.

Performances de la caméra

Les performances de la caméra sont mesurées en tant qu'erreurs systématiques et aléatoires.

Erreur systématique

Une erreur systématique est définie comme la différence entre la profondeur mesurée après la suppression du bruit et la profondeur correcte (réalité du terrain). Nous calculons la moyenne temporelle sur de nombreuses images d'une scène statique pour éliminer autant que possible le bruit de profondeur. Plus précisément, l'erreur systématique est définie comme suit :

$$E_{systematic} = \frac{\sum_{t=1}^N d_t}{N} - d_{gt}$$

Où d_t indique la profondeur de mesure au moment t , N le nombre d'images utilisées dans la procédure de calcul de moyenne, et d_{gt} la profondeur correspondant à la réalité du terrain.

La spécification d'erreur systématique de la caméra de profondeur exclut l'interférence multichemin (MPI). Une interférence multichemin se produit quand un pixel de capteur intègre la lumière reflétée par plusieurs objets. Elle est en partie atténuée dans notre caméra de profondeur à l'aide de fréquences de modulation supérieures, ainsi que de l'invalidation de profondeur que nous présenterons plus tard.

Erreur aléatoire

Supposons que nous prenons 100 images du même objet sans déplacer la caméra. La profondeur de l'objet sera légèrement différente dans chacune des images. Cette différence est due au bruit de grenaille. Le bruit de grenaille est dû au fait que le nombre de photons qui atteignent le capteur varie dans une proportion aléatoire au fil

du temps. Nous définissons cette erreur aléatoire sur une scène statique comme l'écart type de profondeur dans le temps calculé comme suit :

$$E_{random} = \sqrt{\frac{\sum_{t=1}^N (d_t - \bar{d})^2}{N}}$$

Où N indique le nombre de mesures de profondeur, d_t la mesure de profondeur au moment t , et \bar{d} la valeur moyenne calculée sur toutes les mesures de profondeur d_t .

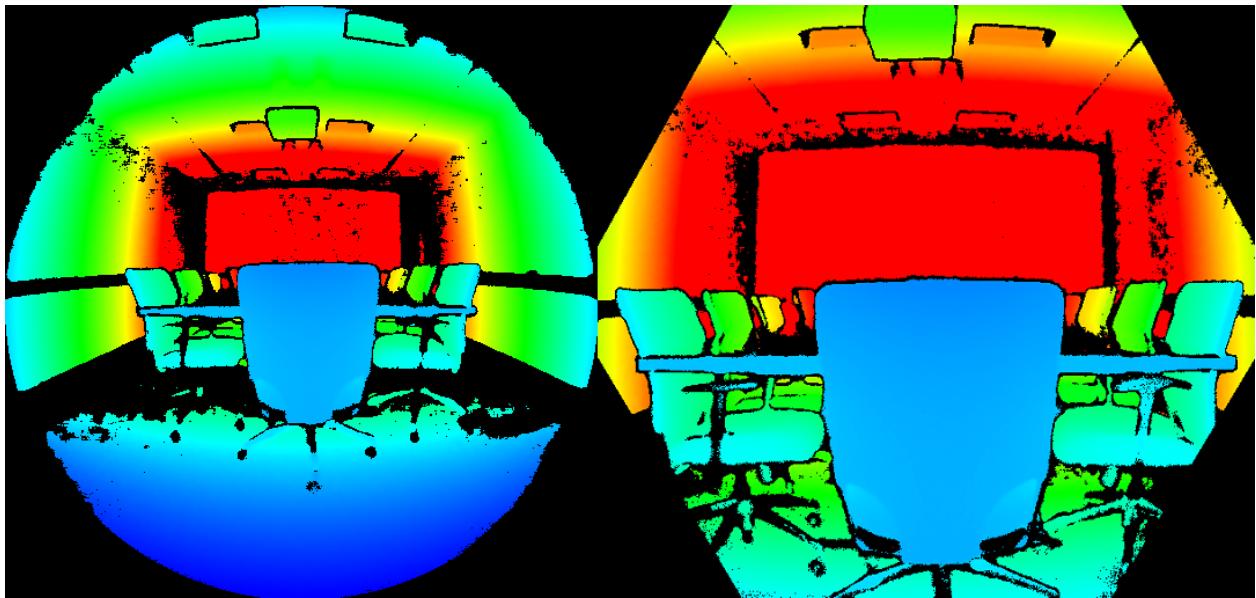
Invalidation

Dans certains cas, la caméra de profondeur ne peut pas fournir de valeurs correctes pour certains pixels. Les pixels de profondeur sont alors invalidés. Les pixels non valides sont indiqués par une valeur de profondeur égale à 0. Les raisons pour lesquelles le moteur de profondeur ne peut pas produire des valeurs correctes sont les suivantes :

- Hors du masque d'illumination IR actif
- Signal IR saturé
- Signal IR faible
- Valeur hors norme de filtrer
- Interférence multichemin

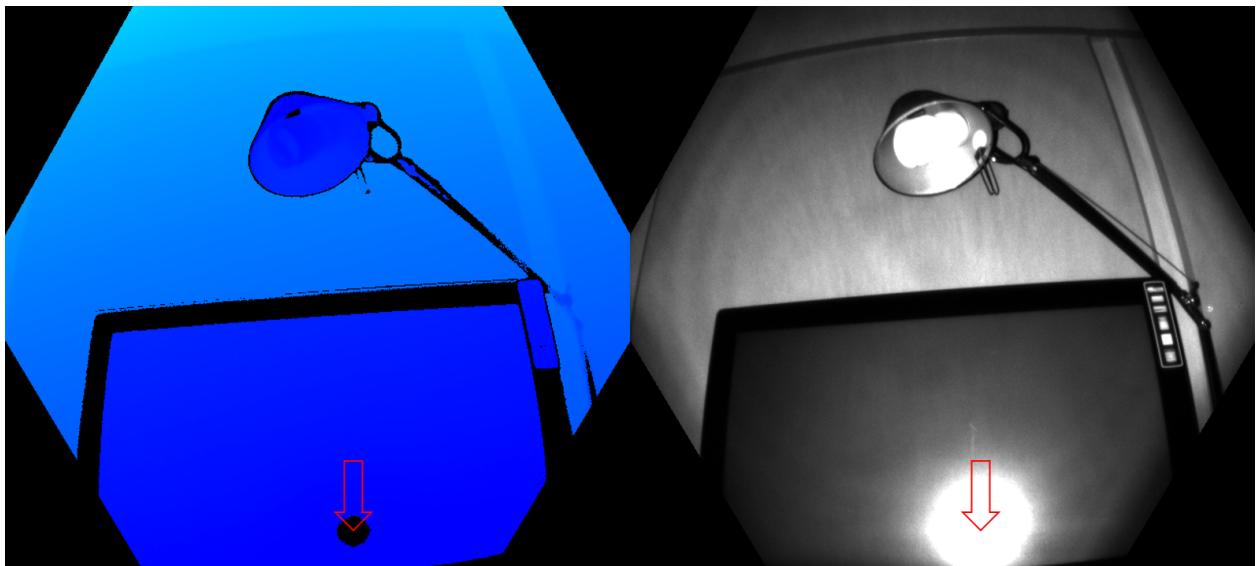
Masque d'illumination

Des pixels sont invalidés quand ils sont hors du masque d'illumination IR actif. Nous vous déconseillons d'utiliser le signal de tels pixels pour calculer la profondeur. La figure ci-dessous illustre l'exemple d'invalidation par masque d'illumination. Les pixels invalidés sont ceux de couleur noire situés hors du cercle dans les modes FoV large (à gauche), et de l'hexagone dans les FoV étroit (à droite).

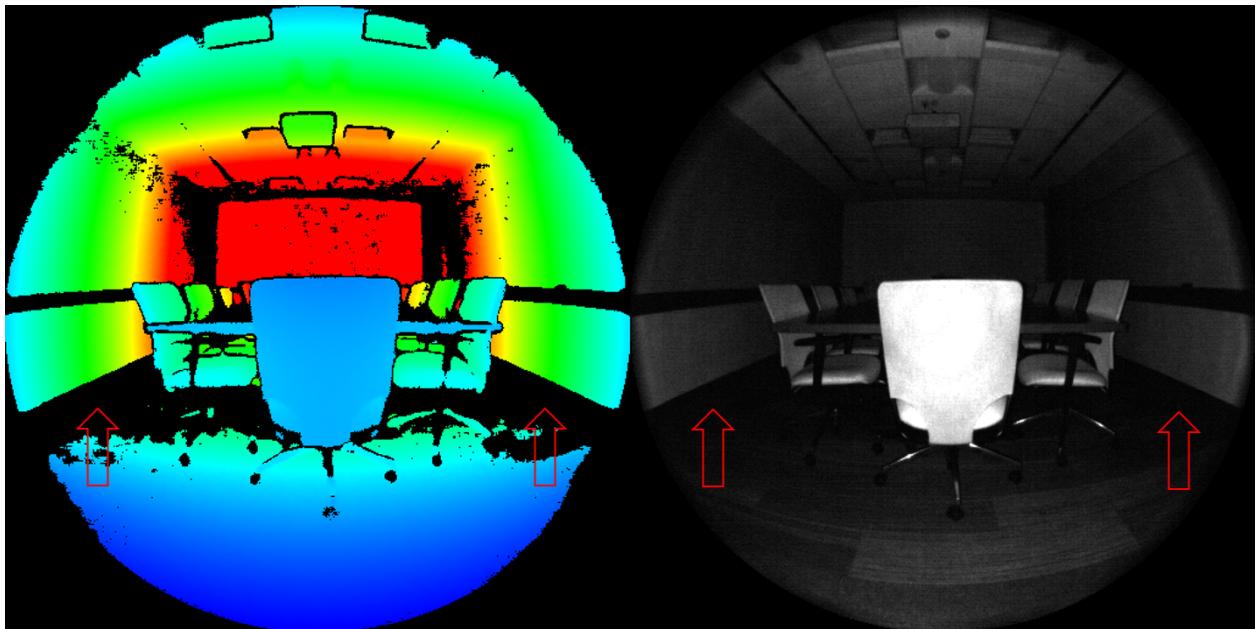


Force du signal

Des pixels sont invalidés quand ils contiennent un signal IR saturé. Quand les pixels sont saturés, les informations de phase sont perdues. L'image ci-dessous montre l'exemple d'invalidation d'un signal IR saturé. Consultez les flèches pointant vers les exemples de pixels dans les images de profondeur et IR.



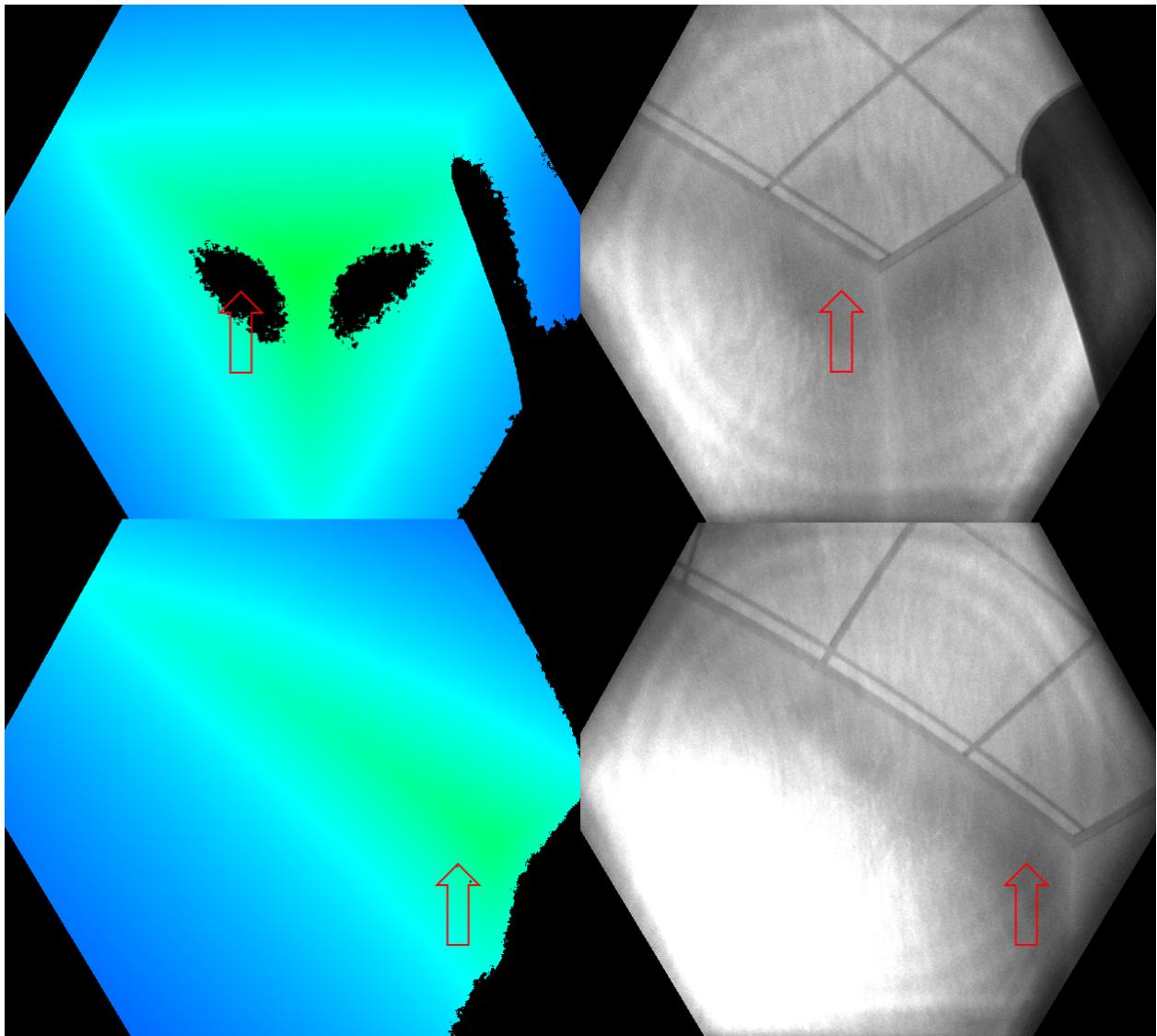
Une invalidation peut également se produire lorsque le signal IR n'est pas suffisamment fort pour générer une profondeur. La figure ci-dessous illustre l'exemple d'invalidation par un signal IR faible. Consultez les flèches pointant vers des exemples de pixels dans les images de profondeur et IR.



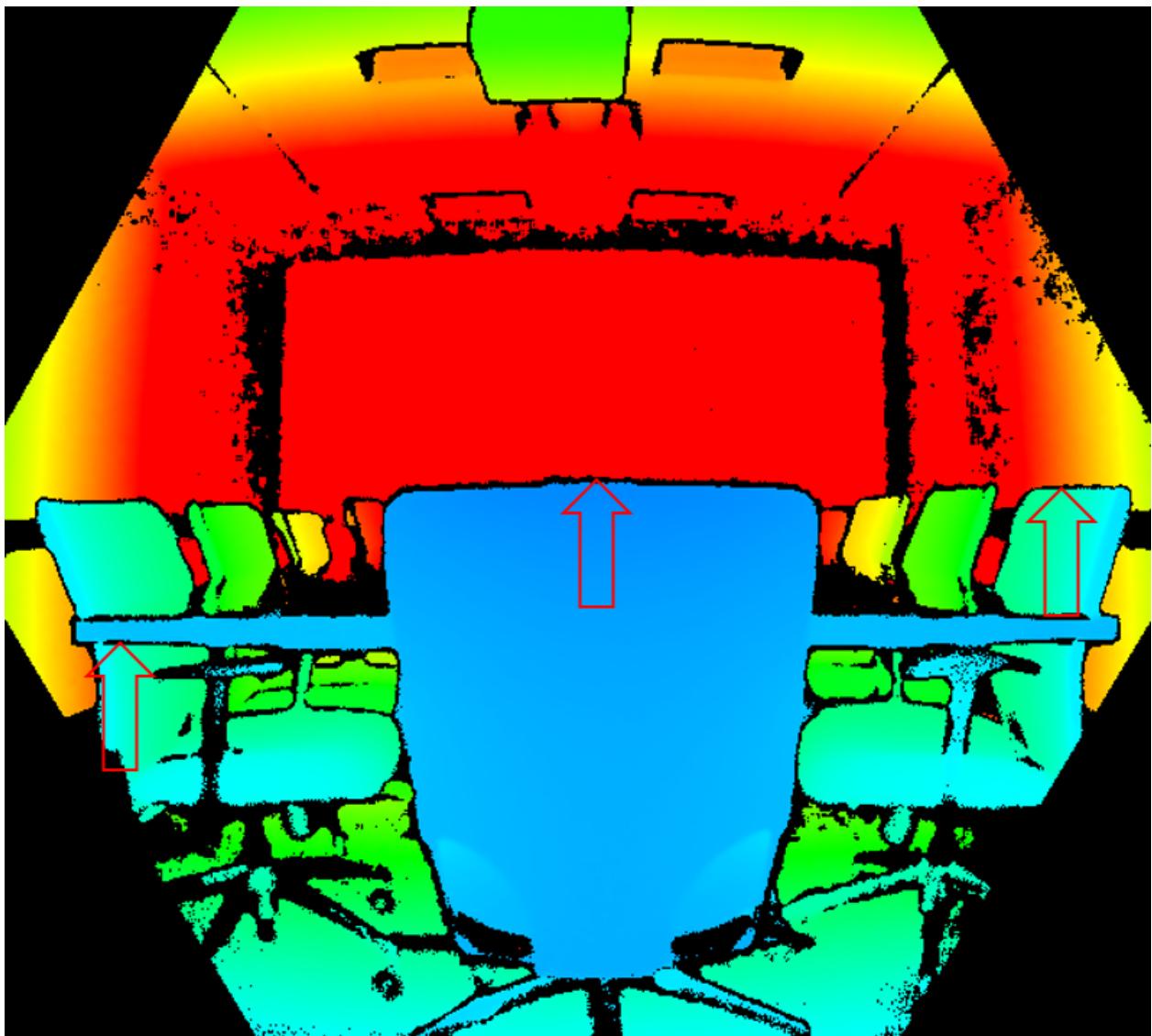
Profondeur ambiguë

Des pixels peuvent également être invalidés s'ils ont reçu des signaux de plusieurs objets dans la scène. Un cas courant où ce type d'invalidation peut être constaté est celui des coins. En raison de la géométrie de la scène, la lumière IR de la caméra se réfléchit d'un mur sur l'autre. Cette lumière réfléchie entraîne une ambiguïté dans la profondeur mesurée du pixel. Les filtres de l'algorithme de profondeur détectent ces signaux ambigus et invalident les pixels.

Les figures ci-dessous présentent des exemples d'invalidation par détection multichemin. Vous pouvez également voir comment la surface d'exposition invalidée dans une vue de caméra (ligne supérieure) peut réapparaître dans une autre vue caméra (ligne inférieure). Cette image montre que les surfaces invalidées dans une perspective peuvent être visibles dans une autre.



Un autre cas courant d'invalidation multichemin est quand des pixels qui contiennent un signal mixte pour le premier plan et l'arrière-plan (par exemple autour des bords d'objets). En cas de mouvement rapide, vous pouvez constater que davantage de pixels sont invalidés autour des bords. Les pixels supplémentaires sont invalidés en raison de l'intervalle d'exposition de la capture de profondeur brute.



Étapes suivantes

Systèmes de coordonnées

Systèmes de coordonnées Azure Kinect

DK

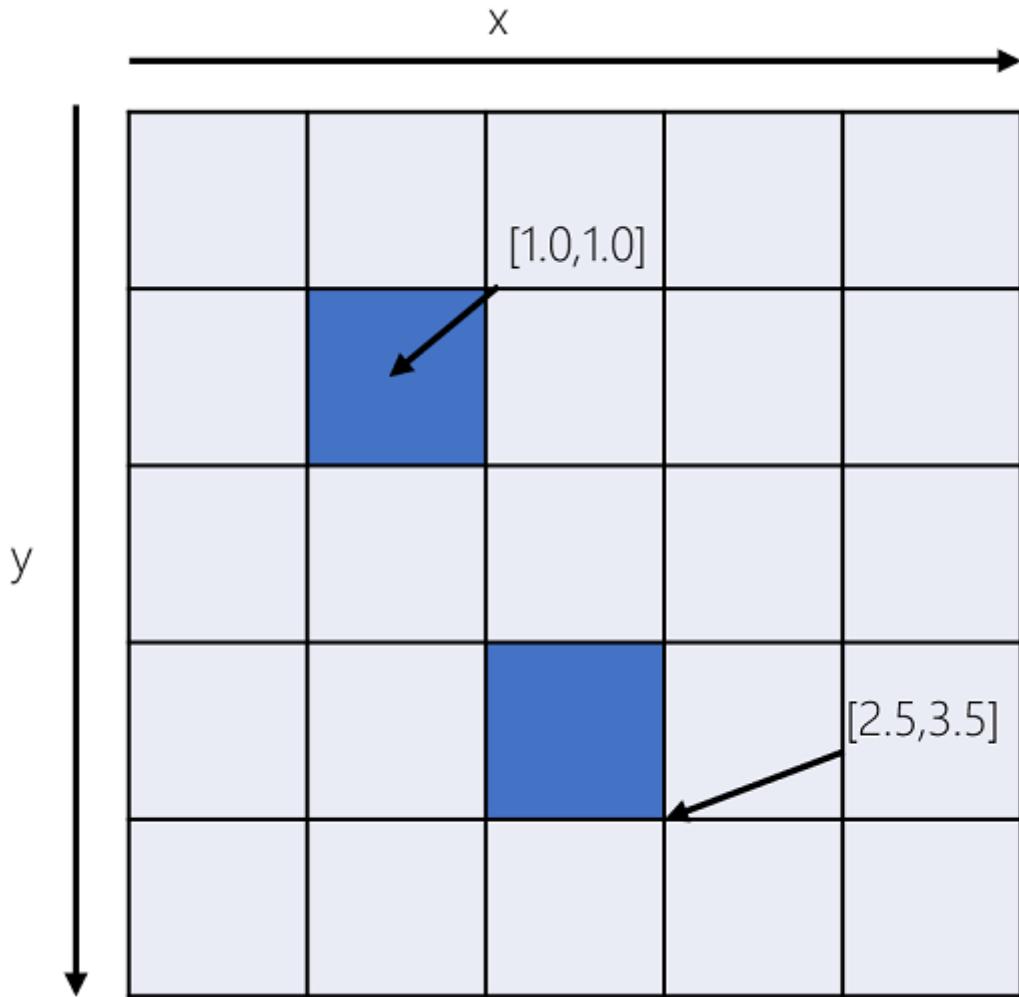
Article • 01/06/2023

Dans cet article, nous décrivons les conventions utilisées pour les systèmes de coordonnées 2D et 3D. Il existe des systèmes de coordonnées distincts associés à chaque capteur d'appareil et aux [fonctions d'étalonnage](#) autorisées à convertir des points entre eux. Les [fonctions de transformation](#) convertissent des images entières entre systèmes de coordonnées.

Systèmes de coordonnées 2D

Les caméras de profondeur et couleur sont associées à un système de coordonnées 2D indépendant. Une coordonnée $[x,y]$ est représentée en unités de pixels où x est compris entre 0 et largeur-1 et y entre 0 et hauteur-1. La largeur et la hauteur dépendent du choix du mode dans lequel les caméras de profondeur et couleur sont utilisées. La coordonnée de pixel $[0,0]$ correspond au pixel supérieur gauche de l'image. Des coordonnées de pixels peuvent être fractionnelles et représenter des coordonnées de sous-pixel.

Le système de coordonnées 2D est centré sur 0. Autrement dit, la coordonnée de sous-pixel $[0.0, 0.0]$ représente le centre, et la coordonnée $[0.5, 0.5]$ l'angle inférieur droit du pixel, comme illustré ci-dessous.



Systèmes de coordonnées 3D

Chaque caméra, l'accéléromètre et le gyroscope sont associés à un système d'espace de coordonnées 3D indépendant.

Les points dans les systèmes de coordonnées 3D sont représentés en tant que triplets de coordonnées [X,Y,Z] de métrique avec des unités exprimées en millimètres.

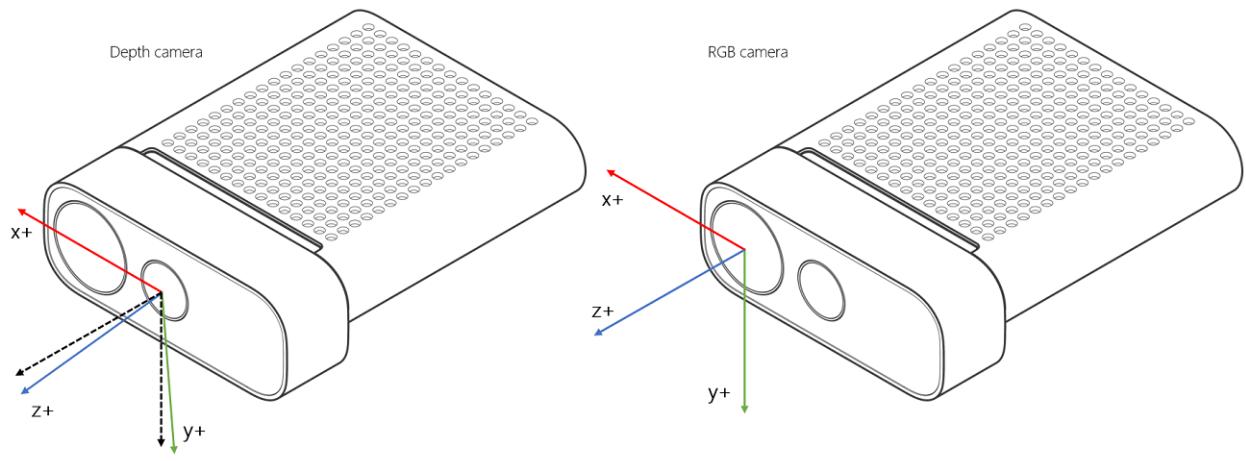
Caméra de profondeur et couleur

L'origine $[0,0,0]$ se trouve au point focal de la caméra. Le système de coordonnées est orienté de manière à ce que l'axe X positif pointe vers la droite, l'axe Y positif vers le bas, et l'axe Z positif vers l'avant.

La caméra de profondeur est inclinée de 6 degrés vers le bas de la caméra de couleur, comme illustré ci-dessous.

La caméra de profondeur utilise deux illuminateurs. L'illuminateur utilisé dans les modes de champ de vision étroit (NFOV, narrow field-of-view) étant aligné avec le boîtier de la

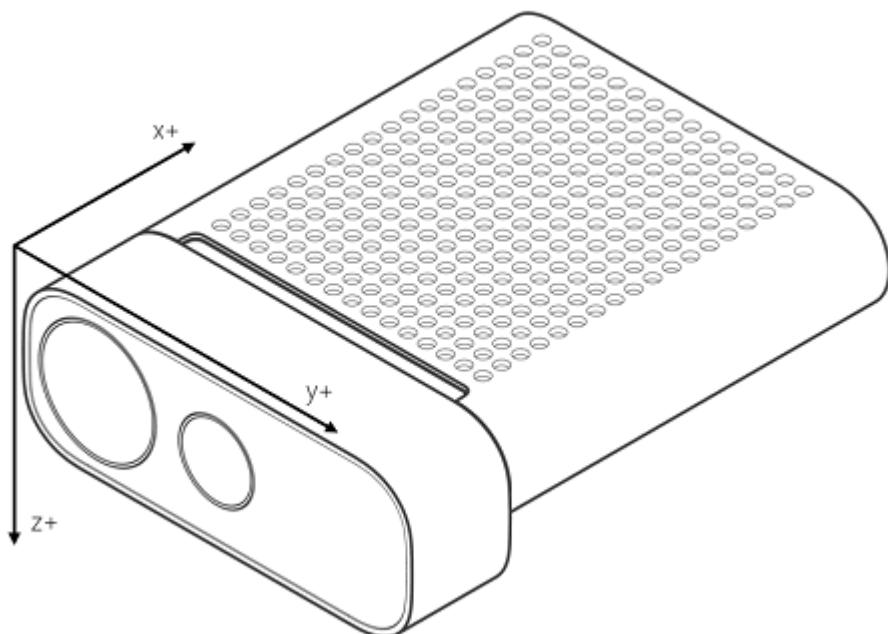
caméra de profondeur, l'illuminateur n'est pas incliné. L'illuminateur utilisé dans les modes de champ de vision large (WFOV, wide field-of-view) est incliné de 1,3 degrés vers le bas par rapport à la caméra de profondeur.



Gyroscope et accéléromètre

L'origine $[0,0,0]$ du gyroscope est identique à celle de la caméra de profondeur.

L'origine de l'accéléromètre coïncide avec son emplacement physique. Les systèmes de coordonnées de l'accéléromètre et du gyroscope sont droitiers. L'axe X positif du système de coordonnées pointe vers l'arrière, l'axe Y positif vers la gauche et l'axe Z positif vers le bas, comme illustré ci-dessous.



Étapes suivantes

Découvrir le Kit de développement logiciel (SDK) de capteur Kinect Azure

Jointures de suivi du corps Azure Kinect

Article • 01/06/2023

Le suivi du corps Azure Kinect peut effectuer le suivi de plusieurs corps humains en même temps. Chaque corps comprend un ID pour la corrélation temporelle entre les images et le squelette cinématique. Le nombre de corps détectés dans chaque image peut être obtenu à l'aide de `k4abt_frame_get_num_bodies()`.

Jointures

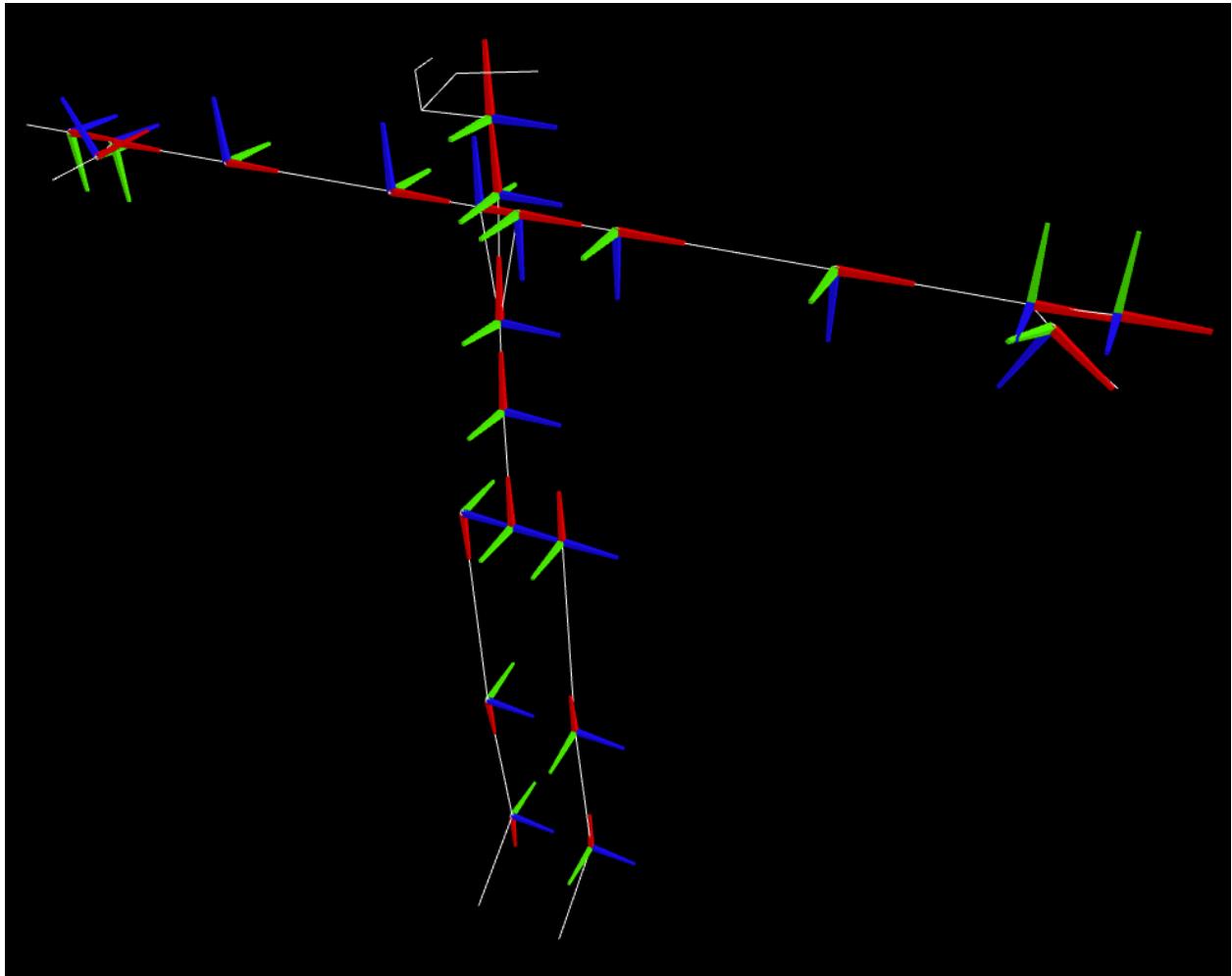
La position et l'orientation de la jointure sont des estimations relatives à l'image de référence du capteur de profondeur global. La position est spécifiée en millimètres. L'orientation est exprimée sous la forme d'un quaternion normalisé.

Coordonnées de jointures

La position et l'orientation de chaque jointure forment le système de coordonnées de jointures de droite. Tous les systèmes de coordonnées de jointures sont des systèmes absolus dans le système de coordonnées 3D de la caméra de profondeur.

ⓘ Notes

Le choix d'une orientation d'axe retourné pour les jointures correspondantes entre les deux côtés du corps vise à simplifier le mouvement miroir, par exemple relever les deux bras de plus de 20 degrés, ce qui est courant pour les avatars commerciaux, les moteurs de jeux et les logiciels de rendu.



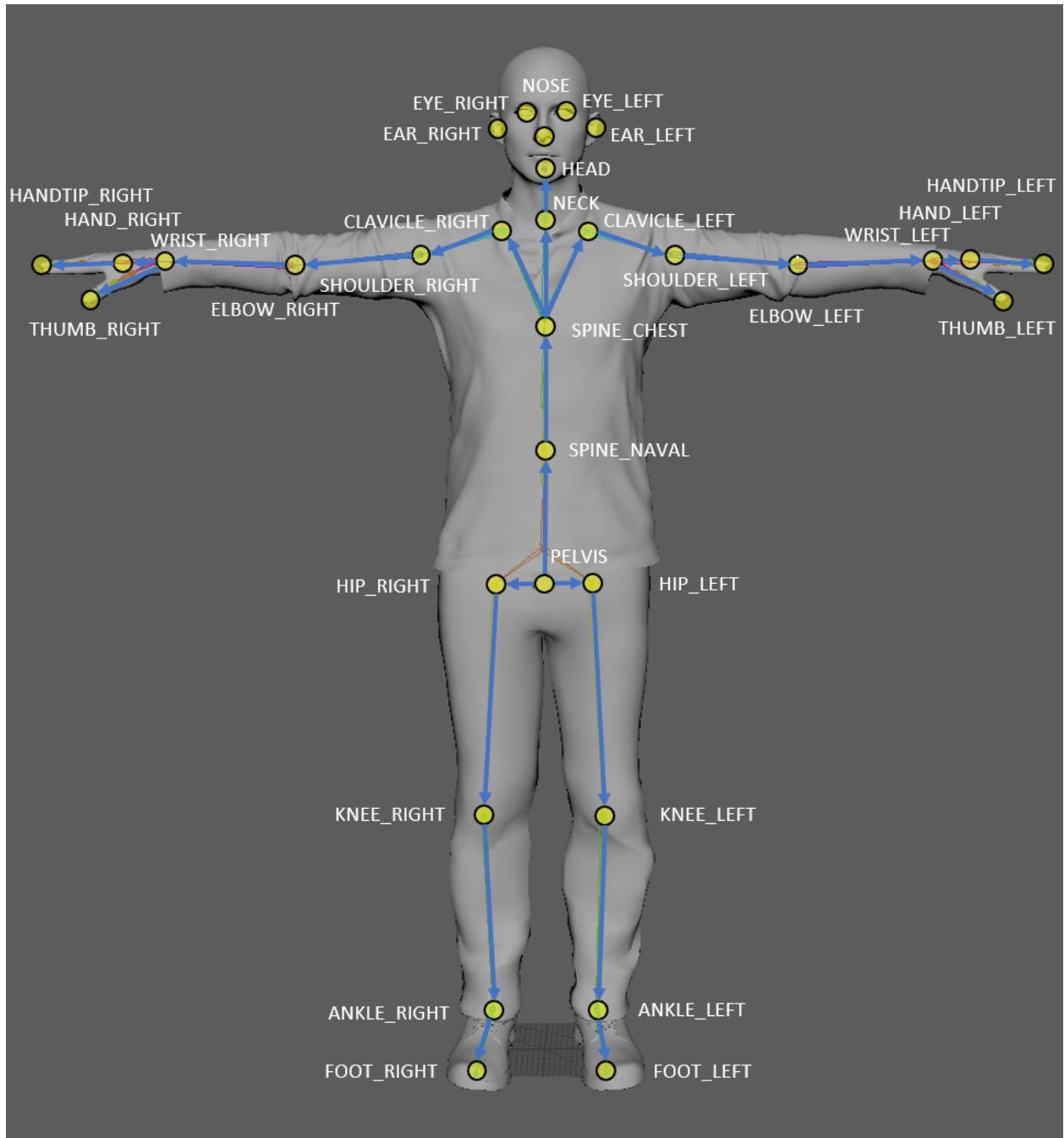
Légende : | axe X = rouge | axe Y = vert | axe Z = bleu |

ⓘ Notes

La sortie visuelle de l'outil `k4abt_simple_3d_viewer.exe` est mise en miroir.

Hiérarchie des jointures

Un squelette compte 32 jointures qui sont hiérarchisées du centre du corps aux extrémités. Chaque connexion (os) lie une jointure parente à une jointure enfant. La figure illustre les emplacements et la connexion des jointures par rapport au corps humain.



Le tableau suivant énumère les connexions de jointures standard.

| Index | Nom de jointure | Jointure parente |
|-------|------------------|------------------|
| 0 | BASSIN | - |
| 1 | COLONNE_NOMBRL | BASSIN |
| 2 | COLONNE_THORAX | COLONNE_NOMBRL |
| 3 | COU | COLONNE_THORAX |
| 4 | CLAVICULE_GAUCHE | COLONNE_THORAX |
| 5 | ÉPAULE_GAUCHE | CLAVICULE_GAUCHE |
| 6 | COUDE_GAUCHE | ÉPAULE_GAUCHE |

| Index | Nom de jointure | Jointure parente |
|--------------|------------------------|-------------------------|
| 7 | POIGNET_GAUCHE | COUDE_GAUCHE |
| 8 | MAIN_GAUCHE | POIGNET_GAUCHE |
| 9 | BOUTMAIN_GAUCHE | MAIN_GAUCHE |
| 10 | POUCE_GAUCHE | POIGNET_GAUCHE |
| 11 | CLAVICULE_DROITE | COLONNE_THORAX |
| 12 | ÉPAULE_DROITE | CLAVICULE_DROITE |
| 13 | COUDE_DROIT | ÉPAULE_DROITE |
| 14 | POIGNET_DROIT | COUDE_DROIT |
| 15 | MAIN_DROITE | POIGNET_DROIT |
| 16 | BOUTMAIN_DROITE | MAIN_DROITE |
| 17 | POUCE_DROIT | POIGNET_DROIT |
| 18 | HANCHE_GAUCHE | BASSIN |
| 19 | GENOU_GAUCHE | HANCHE_GAUCHE |
| 20 | CHEVILLE_GAUCHE | GENOU_GAUCHE |
| 21 | PIED_GAUCHE | CHEVILLE_GAUCHE |
| 22 | HANCHE_DROITE | BASSIN |
| 23 | GENOU_DROIT | HANCHE_DROITE |
| 24 | CHEVILLE_DROITE | GENOU_DROIT |
| 25 | PIED_DROIT | CHEVILLE_DROITE |
| 26 | HEAD | COU |
| 27 | NEZ | TÊTE |
| 28 | ŒIL_GAUCHE | TÊTE |
| 29 | OREILLE_GAUCHE | TÊTE |
| 30 | ŒIL_DROIT | TÊTE |
| 31 | OREILLE_DROITE | TÊTE |

Étapes suivantes

[Mappage d'index de suivi de corps](#)

Mappage d'index de suivi de corps Azure Kinect

Article • 01/06/2023

Le mappage d'index de corps comprend le mappage de segmentation d'instance pour chaque corps dans la capture de la caméra de profondeur. Chaque pixel est mappé au pixel correspondant dans l'image de profondeur ou IR. La valeur de chaque pixel représente le corps auquel le pixel appartient. Il peut s'agir de l'arrière-plan (valeur `K4ABT_BODY_INDEX_MAP_BACKGROUND`) ou de l'index d'un `k4abt_body_t` détecté.



ⓘ Notes

L'index du corps est différent de l'ID du corps. Vous pouvez interroger l'ID du corps à partir d'un index de corps donné en appelant l'API `k4abt_frame_get_body_id()`.

Utilisation du mappage d'index de corps

Le mappage d'index de corps est stocké en tant que `k4a_image_t` et a la même résolution que l'image de profondeur ou IR. Chaque pixel est une valeur de 8 bits. Il est

possible d'interroger celle-ci à partir d'un `k4abt_frame_t` en appelant `k4abt_frame_get_body_index_map`. Il incombe au développeur de libérer de la mémoire pour le mappage d'index de corps en appelant `k4a_image_release()`.

Étapes suivantes

[Créer votre première application de suivi de corps](#)

À propos du Kit de développement logiciel (SDK) du capteur Azure Kinect

Article • 01/06/2023

Cet article fournit une vue d'ensemble du Kit de développement logiciel (SDK) du capteur Azure Kinect, de ses fonctionnalités et de ses outils.

Fonctionnalités

Le Kit de développement logiciel (SDK) du capteur Azure Kinect fournit un accès de bas niveau multiplateforme pour la configuration d'appareil Azure Kinect et les flux de capteurs matériels, à savoir :

- Accès aux caméras de profondeur et contrôle du mode (mode IR passif et modes de profondeur à champ de vision large et étroit)
- Accès aux caméras RVB et contrôle de celles-ci (par exemple, exposition et balance des blancs)
- Accès aux capteurs de mouvement (gyroscope et accéléromètre)
- Diffusion synchronisée des données des caméras RVB de profondeur avec la possibilité de configurer le délai entre les caméras
- Contrôle de synchronisation des appareils externes avec la possibilité de configurer le décalage entre les appareils
- Accès aux métadonnées de vue de caméra pour la résolution d'image, l'horodatage, etc.
- Accès aux données d'étalonnage des appareils

Outils

- Une [visionneuse Azure Kinect](#) permettant de superviser les flux de données des appareils et de configurer les différents modes.
- Un [Enregistreur Azure Kinect](#) et une API de lecture de capteur utilisant le [format de conteneur Matroska](#).
- Un [outil de mise à jour de microprogramme Azure Kinect DK](#).

Kit de développement logiciel (SDK) du capteur

- [Téléchargez le Kit de développement logiciel \(SDK\) du capteur](#).

- Le Kit de développement logiciel (SDK) du capteur est disponible en [open source](#) sur [GitHub](#).
- Pour plus d'informations sur l'utilisation, consultez la [documentation de l'API du Kit de développement logiciel \(SDK\) du capteur](#).

Étapes suivantes

Maintenant que vous avez découvert le kit de développement logiciel (SDK) du capteur Azure Kinect, vous pouvez également :

[Télécharger le code du Kit de développement logiciel \(SDK\) du capteur](#)

[Rechercher et ouvrir un appareil](#)

Rechercher et ouvrir l'appareil Azure Kinect

Article • 01/06/2023

Cet article décrit comment trouver, puis ouvrir votre Azure Kinect DK. Il article explique comment gérer le cas où plusieurs appareils sont connectés à votre ordinateur.

Vous pouvez également consulter l'[exemple d'énumération du Kit de développement logiciel \(SDK\)](#) qui montre comment utiliser les fonctions décrites dans cet article.

Voici les fonctions qui sont abordées ici :

- [k4a_device_get_installed_count\(\)](#)
- [k4a_device_open\(\)](#)
- [k4a_device_get_serialnum\(\)](#)
- [k4a_device_close\(\)](#)

Découvrir le nombre d'appareils connectés

Commencez par récupérer le nombre d'appareils Azure Kinect actuellement connectés en utilisant [k4a_device_get_installed_count\(\)](#).

C

```
uint32_t device_count = k4a_device_get_installed_count();

printf("Found %d connected devices:\n", device_count);
```

Ouvrir un appareil

Pour obtenir des informations sur un appareil ou pour y lire des données, vous devez d'abord ouvrir un descripteur de l'appareil en utilisant [k4a_device_open\(\)](#).

C

```
k4a_device_t device = NULL;

for (uint8_t deviceIndex = 0; deviceIndex < device_count; deviceIndex++)
{
    if (K4A_RESULT_SUCCEEDED != k4a_device_open(deviceIndex, &device))
    {
        printf("%d: Failed to open device\n", deviceIndex);
```

```
        continue;
    }

    ...

    k4a_device_close(device);
}
```

Le paramètre `index` de [k4a_device_open\(\)](#) indique l'appareil à ouvrir si plusieurs appareils sont connectés. Si vous attendez à ce qu'un seul appareil soit connecté, vous pouvez passer un argument `K4A_DEVICE_DEFAULT` ou 0 pour indiquer le premier appareil.

Chaque fois que vous ouvrez un appareil, vous devez appeler [k4a_device_close\(\)](#) lorsque vous avez fini d'utiliser le descripteur. Vous ne pouvez ouvrir aucun autre descripteur sur le même appareil tant que vous n'avez pas fermé le descripteur ouvert.

Identifier un appareil spécifique

L'ordre d'énumération des appareils par `index` ne change pas tant que vous ne connectez ou ne déconnectez pas un appareil. Pour identifier un appareil physique, vous devez utiliser son numéro de série.

Pour lire le numéro de série de l'appareil, utilisez la fonction [k4a_device_get_serialnum\(\)](#) après avoir ouvert un descripteur.

Cet exemple montre comment allouer la quantité de mémoire appropriée pour stocker le numéro de série.

```
C

char *serial_number = NULL;
size_t serial_number_length = 0;

if (K4A_BUFFER_RESULT_TOO_SMALL != k4a_device_get_serialnum(device, NULL,
&serial_number_length))
{
    printf("%d: Failed to get serial number length\n", deviceIndex);
    k4a_device_close(device);
    device = NULL;
    continue;
}

serial_number = malloc(serial_number_length);
if (serial_number == NULL)
{
    printf("%d: Failed to allocate memory for serial number (%zu bytes)\n",
deviceIndex,
```

```

deviceIndex, serial_number_length);
    k4a_device_close(device);
    device = NULL;
    continue;
}

if (K4A_BUFFER_RESULT_SUCCEEDED != k4a_device_get_serialnum(device,
serial_number, &serial_number_length))
{
    printf("%d: Failed to get serial number\n", deviceIndex);
    free(serial_number);
    serial_number = NULL;
    k4a_device_close(device);
    device = NULL;
    continue;
}

printf("%d: Device \"%s\"\n", deviceIndex, serial_number);

```

Ouvrir l'appareil par défaut

Dans la plupart des applications, il n'y a qu'un seul appareil Azure Kinect DK connecté au même ordinateur. Si vous avez uniquement besoin de vous connecter au seul appareil attendu, vous pouvez appeler la fonction [k4a_device_open\(\)](#) avec l'`index` de `K4A_DEVICE_DEFAULT` pour ouvrir le premier appareil.

C

```

k4a_device_t device = NULL;
uint32_t device_count = k4a_device_get_installed_count();

if (device_count != 1)
{
    printf("Unexpected number of devices found (%d)\n", device_count);
    goto Exit;
}

if (K4A_RESULT_SUCCEEDED != k4a_device_open(K4A_DEVICE_DEFAULT, &device))
{
    printf("Failed to open device\n");
    goto Exit;
}

```

Étapes suivantes

[Récupérer les images](#)

Récupérer des exemples IMU

Récupérer des données d'image Azure Kinect

Article • 01/06/2023

Cette page fournit des informations détaillées sur la façon de récupérer des images à partir de l'appareil Azure Kinect. L'article montre comment capturer des images coordonnées et y accéder entre la couleur et la profondeur de l'appareil. Pour accéder aux images, vous devez commencer par ouvrir et configurer l'appareil. Vous pouvez ensuite capturer des images. Avant de configurer et de capturer une image, vous devez [Rechercher et ouvrir un appareil](#).

Vous pouvez également consulter [l'exemple de diffusion en continu du Kit de développement logiciel \(SDK\)](#) qui montre comment utiliser les fonctions abordées dans cet article.

Voici les fonctions qui sont abordées ici :

- [k4a_device_start_cameras\(\)](#)
- [k4a_device_get_capture\(\)](#)
- [k4a_capture_get_depth_image\(\)](#)
- [k4a_image_get_buffer\(\)](#)
- [k4a_image_release\(\)](#)
- [k4a_capture_release\(\)](#)
- [k4a_device_stop_cameras\(\)](#)

Configurer et démarrer l'appareil

Les deux caméras disponibles sur votre appareil Kinect prennent en charge plusieurs modes, résolutions et formats de sortie. Pour obtenir une liste complète, reportez-vous au Kit de développement Azure Kinect [Spécifications matérielles](#).

La configuration de la diffusion en continu est définie à l'aide de valeurs de la structure [k4a_device_configuration_t](#).

C

```
k4a_device_configuration_t config = K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;  
config.camera_fps = K4A_FRAMES_PER_SECOND_30;  
config.color_format = K4A_IMAGE_FORMAT_COLOR_MJPG;  
config.color_resolution = K4A_COLOR_RESOLUTION_2160P;  
config.depth_mode = K4A_DEPTH_MODE_NFOV_UNBINNED;
```

```
if (K4A_RESULT_SUCCEEDED != k4a_device_start_cameras(device, &config))
{
    printf("Failed to start device\n");
    goto Exit;
}
```

Une fois les caméras démarrées, elles continuent à capturer des données jusqu'à l'appel de la fonction [k4a_device_stop_cameras\(\)](#) ou à la fermeture de l'appareil.

Stabilisation

Lors du démarrage d'appareils à l'aide de la fonctionnalité de synchronisation d'appareils multiples, il est fortement recommandé de le faire à l'aide d'un paramètre à exposition fixe. Avec une exposition manuelle, jusqu'à huit captures de l'appareil peuvent être nécessaires avant la stabilisation des images et des taux de trames. Avec une exposition automatique, jusqu'à 20 captures avant que peuvent être nécessaires avant la stabilisation des images et des taux de trames.

Obtenir une capture à partir de l'appareil

Les images sont capturées à partir de l'appareil de manière corrélée. Chaque image capturée contient une image de profondeur, une image IR, une image en couleur ou une combinaison d'images.

Par défaut, l'API retourne une capture uniquement une fois qu'elle a reçu toutes les images demandées pour le mode de diffusion en continu. Vous pouvez configurer l'API de sorte qu'elle retourne des captures partielles avec uniquement des images de profondeur ou en couleurs dès qu'elles sont disponibles en désactivant le paramètre [synchronized_images_only](#) de la fonction [k4a_device_configuration_t](#).

C

```
// Capture a depth frame
switch (k4a_device_get_capture(device, &capture, TIMEOUT_IN_MS))
{
    case K4A_WAIT_RESULT_SUCCEEDED:
        break;
    case K4A_WAIT_RESULT_TIMEOUT:
        printf("Timed out waiting for a capture\n");
        continue;
        break;
    case K4A_WAIT_RESULT_FAILED:
        printf("Failed to read a capture\n");
        goto Exit;
}
```

Une fois que l'API a correctement retourné une capture, vous devez appeler [k4a_capture_release\(\)](#) lorsque vous avez fini d'utiliser l'objet de capture.

Obtenir une image de la capture

Pour récupérer une image capturée, appelez la fonction appropriée pour chaque type d'image. Valeurs possibles :

- [k4a_capture_get_color_image\(\)](#)
- [k4a_capture_get_depth_image\(\)](#)
- [k4a_capture_get_ir_image\(\)](#)

Une fois que vous avez fini d'utiliser l'image, vous devez appeler la fonction [k4a_image_release\(\)](#) sur tout descripteur [k4a_image_t](#) retourné par ces fonctions.

Accéder aux tampons d'image

[k4a_image_t](#) possède de nombreuses fonctions d'accesseur pour l'extraction de propriétés de l'image.

Pour accéder à la mémoire tampon de l'image, utilisez [k4a_image_get_buffer](#).

L'exemple suivant montre comment accéder à une image de profondeur capturée. Ce même principe s'applique à d'autres types d'images. Toutefois, veillez à remplacer la variable de type image par le type d'image correct, tel que IR ou Couleur.

C

```
// Access the depth16 image
k4a_image_t image = k4a_capture_get_depth_image(capture);
if (image != NULL)
{
    printf(" | Depth16 res:%4dx%4d stride:%5d\n",
           k4a_image_get_height_pixels(image),
           k4a_image_get_width_pixels(image),
           k4a_image_get_stride_bytes(image));

    // Release the image
    k4a_image_release(image);
}

// Release the capture
k4a_capture_release(capture);
```

Étapes suivantes

Vous savez maintenant comment capturer et coordonner les images des caméras couleur et de profondeur à l'aide de votre appareil Azure Kinect. Vous pouvez également :

[Récupérer des échantillons IMU](#)

[Accéder aux microphones](#)

Récupérer des échantillons d'IMU Azure Kinect

Article • 01/06/2023

L'appareil Azure Kinect donne accès aux unités de mesure inertielle (IMU), notamment des types accéléromètre et gyroscope. Pour accéder aux échantillons d'IMU, vous devez ouvrir et configurer votre appareil, puis capturer des données d'IMU. Pour plus d'informations, consultez [Rechercher et ouvrir un appareil](#).

Les échantillons d'IMU sont générés à une fréquence sensiblement plus élevée que les images. Les échantillons sont signalés à l'hôte à un débit inférieur à celui de leur collecte. Lors de l'attente d'un échantillon d'IMU, il est fréquent que plusieurs échantillons deviennent disponibles en même temps.

Pour plus d'informations sur le taux de création de rapports sur les IMU, consultez les [spécifications matérielles](#) d'Azure Kinect DK.

Configurer et démarrer des caméras

ⓘ Notes

Les capteurs IMU ne peuvent fonctionner que lorsque les caméras couleur et/ou de profondeur sont en cours d'exécution. Les capteurs IMU ne peuvent pas fonctionner isolément.

Pour démarrer les caméras, utilisez [k4a_device_start_cameras\(\)](#).

C

```
k4a_device_configuration_t config = K4A_DEVICE_CONFIG_INIT_DISABLE_ALL;
config.camera_fps = K4A_FRAMES_PER_SECOND_30;
config.color_format = K4A_IMAGE_FORMAT_COLOR_MJPG;
config.color_resolution = K4A_COLOR_RESOLUTION_2160P;

if (K4A_RESULT_SUCCEEDED != k4a_device_start_cameras(device, &config))
{
    printf("Failed to start cameras\n");
    goto Exit;
}

if (K4A_RESULT_SUCCEEDED != k4a_device_start_imu(device))
{
    printf("Failed to start imu\n");
```

```
    goto Exit;  
}
```

Accéder aux échantillons d'IMU

Chaque échantillon [k4a_imu_sample_t](#) contient des lectures d'accéléromètre et de gyroscope capturées presque en même temps.

Vous pouvez recevoir les échantillons d'IMU sur le même thread que celui sur lequel vous recevez les captures d'images ou sur des threads distincts.

Pour récupérer les échantillons d'IMU dès qu'ils sont disponibles, vous pouvez appeler [k4a_device_get_imu_sample\(\)](#) sur son propre thread. L'API dispose également d'une mise en file d'attente interne suffisante pour vous permettre de vérifier les échantillons uniquement après le retour de chaque capture d'image.

Étant donné qu'il existe une mise en file d'attente interne des échantillons d'IMU, vous pouvez utiliser le modèle suivant sans supprimer de données :

1. Attendez une capture, quelle que soit la fréquence des images.
2. Traitez la capture.
3. Récupérez tous les échantillons d'IMU mis en file d'attente.
4. Répétez l'attente pour la capture suivante.

Pour récupérer tous les échantillons d'IMU actuellement en file d'attente, vous pouvez appeler la fonction [k4a_device_get_imu_sample\(\)](#) avec un `timeout_in_ms` de 0 dans une boucle jusqu'à ce que la fonction retourne `K4A_WAIT_RESULT_TIMEOUT`.

`K4A_WAIT_RESULT_TIMEOUT` indique qu'aucun échantillon n'est en file d'attente et n'est arrivé dans le délai spécifié.

Exemple d'utilisation

```
C  
  
k4a_imu_sample_t imu_sample;  
  
// Capture a imu sample  
switch (k4a_device_get_imu_sample(device, &imu_sample, TIMEOUT_IN_MS))  
{  
case K4A_WAIT_RESULT_SUCCEEDED:  
    break;  
case K4A_WAIT_RESULT_TIMEOUT:  
    printf("Timed out waiting for a imu sample\n");  
    continue;
```

```
        break;
    case K4A_WAIT_RESULT_FAILED:
        printf("Failed to read a imu sample\n");
        goto Exit;
}

// Access the accelerometer readings
if (imu_sample != NULL)
{
    printf(" | Accelerometer temperature:%.2f x:%.4f y:%.4f z: %.4f\n",
           imu_sample.temperature,
           imu_sample.acc_sample.xyz.x,
           imu_sample.acc_sample.xyz.y,
           imu_sample.acc_sample.xyz.z);
}
```

Étapes suivantes

Maintenant que vous savez comment utiliser les échantillons d'IMU, vous pouvez également

[Accéder aux données d'entrée du microphone](#)

Accéder aux données d'entrée du microphone Azure Kinect DK

Article • 01/06/2023

Les [Démarrages rapides du kit de développement logiciel \(SDK\) Speech](#) proposent des exemples d'utilisation du réseau de microphones Azure Kinect DK dans différents langages de programmation. Pour voir un exemple, consultez le démarrage rapide [Reconnaissance vocale en C++ sur Windows à l'aide du Kit de développement logiciel \(SDK\) Speech](#). Le code est disponible [sur GitHub ↗](#).

Accédez au réseau de microphones également via l'API Windows. Pour plus de détails extraits de la documentation de Windows, consultez les documents suivants :

- [Architecture Audio Windows](#)
- [Documentation Windows.Media.Capture](#)
- [Didacticiel pour la capture de Webcam](#)
- [Informations sur l'audio USB](#)

Vous pouvez également consulter la [spécification matérielle du réseau de microphones](#).

Étapes suivantes

[Kit de développement logiciel \(SDK\) Services Speech](#)

Utiliser les transformations d'image du Kit de développement logiciel (SDK) du capteur Azure Kinect

Article • 01/06/2023

Suivez les fonctions spécifiques pour l'utilisation et la transformation d'images entre systèmes de caméras coordonnés dans Azure Kinect DK.

Fonctions k4a_transformation

Toutes les fonctions dont le nom commence par le préfixe *k4a_transformation* opèrent sur des images entières. Elles requièrent le descripteur de transformation [k4a_transformation_t](#) obtenu via la fonction [k4a_transformation_create\(\)](#), et sont non allouées via la fonction [k4a_transformation_destroy\(\)](#). Vous pouvez également consulter dans cette rubrique l'[exemple de transformation](#) du Kit de développement logiciel (SDK) qui montre comment utiliser les trois fonctions.

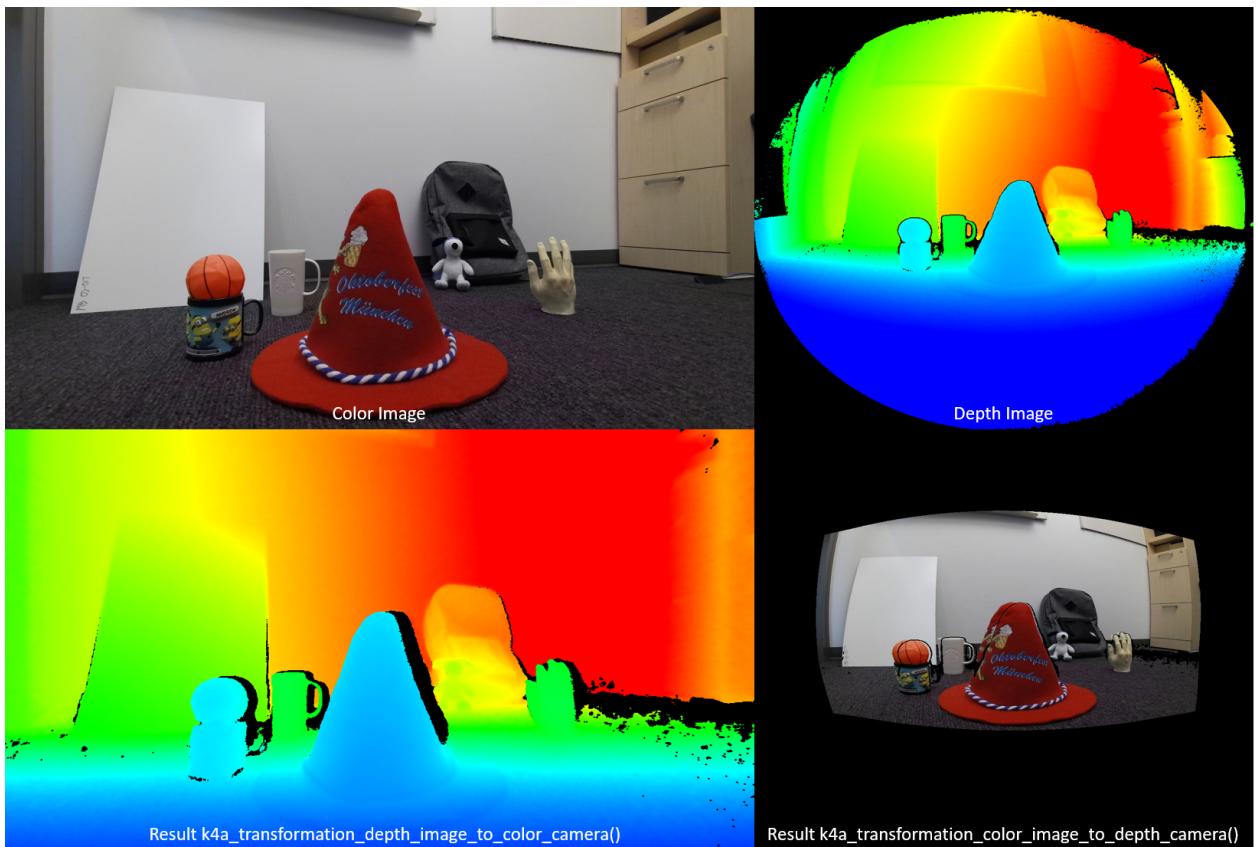
Voici les fonctions qui sont abordées ici :

- [k4a_transformation_depth_image_to_color_camera\(\)](#)
- [k4a_transformation_depth_image_to_color_camera_custom\(\)](#)
- [k4a_transformation_color_image_to_depth_camera\(\)](#)
- [k4a_transformation_depth_image_to_point_cloud\(\)](#)

k4a_transformation_depth_image_to_color_camera

Vue d'ensemble

La fonction [k4a_transformation_depth_image_to_color_camera\(\)](#) transforme la carte de profondeur du point de vue de la caméra de profondeur en point de vue de la caméra couleur. Cette fonction est conçue pour produire des images RVB-D, où D représente un canal d'image supplémentaire qui enregistre la valeur de profondeur. Comme illustré ci-dessous, l'image en couleurs et la sortie de la fonction [k4a_transformation_depth_image_to_color_camera\(\)](#) semblent avoir été prises du même point de vue, c'est-à-dire celui de la caméra couleur.



Implémentation

Cette fonction de transformation est plus complexe que le simple appel de la fonction [k4a_calibration_2d_to_2d\(\)](#) pour chaque pixel. Elle distord le maillage triangulaire de la géométrie de la caméra de profondeur pour l'adapter à la géométrie de la caméra couleur. Le maillage triangulaire est utilisé pour éviter de générer des trous dans l'image de profondeur transformée. Un tampon z-buffer garantit que les occlusions sont gérées correctement. Par défaut , l'accélération GPU est activée pour cette fonction.

Paramètres

Les paramètres d'entrée sont le descripteur de transformation et une image de profondeur. La résolution de l'image de profondeur doit correspondre au `depth_mode` spécifié lors de la création du descripteur de transformation. Par exemple, si le descripteur de transformation a été créé à l'aide du mode `K4A_DEPTH_MODE_WFOV_UNBINNED` 1024x1024, la résolution de l'image de profondeur doit être de 1024 x 1024 pixels. La sortie est une image de profondeur transformée qui doit être allouée par l'utilisateur via l'appel de [k4a_image_create\(\)](#). La résolution de l'image de profondeur transformée doit correspondre à la `color_resolution` spécifiée lors de la création du descripteur de transformation. Par exemple, si la résolution des couleurs a été définie sur `K4A_COLOR_RESOLUTION_1080P`, la résolution de l'image de sortie doit être

de 1920 x 1080 pixels. Le stride de l'image de sortie est défini sur `width * sizeof(uint16_t)`, car l'image stocke des valeurs de profondeur de 16 bits.

k4a_transformation_depth_image_to_color_camera_custom

Vue d'ensemble

La fonction [k4a_transformation_depth_image_to_color_camera_custom\(\)](#) transforme la carte de profondeur et une image personnalisée du point de vue de la caméra de profondeur en point de vue de la caméra couleur. En tant qu'extension de la fonction [k4a_transformation_depth_image_to_color_camera\(\)](#), cette fonction est conçue pour produire une image personnalisée dont chaque pixel correspond à des coordonnées de pixel de la caméra couleur, en plus de l'image de profondeur transformée.

Implémentation

Cette fonction de transformation produit l'image de profondeur transformée de la même façon que la fonction [k4a_transformation_depth_image_to_color_camera\(\)](#). Pour transformer l'image personnalisée, cette fonction fournit des options d'interpolation linéaire ou d'interpolation du voisin le plus proche. Une interpolation linéaire pourrait créer de nouvelles valeurs dans l'image personnalisée transformée. Une interpolation du voisin le plus proche empêche l'affichage dans l'image de sortie de valeurs non présentes dans l'image d'origine, mais produit une image moins lisse. L'image personnalisée doit être un canal unique 8 bits ou 16 bits. Par défaut, l'accélération GPU est activée pour cette fonction.

Paramètres

Les paramètres d'entrée sont le descripteur de transformation, une image de profondeur, une image personnalisée et le type d'interpolation. Les résolutions de l'image de profondeur de l'image personnalisée doivent correspondre au `depth_mode` spécifié lors de la création du descripteur de transformation. Par exemple, si le descripteur de transformation a été créé à l'aide du mode `K4A_DEPTH_MODE_WFOV_UNBINNED` 1024x1024, les résolutions de l'image de profondeur et de l'image personnalisée doivent être de 1024 x 1024 pixels. Le `interpolation_type` doit être `K4A_TRANSFORMATION_INTERPOLATION_TYPE_LINEAR` ou `K4A_TRANSFORMATION_INTERPOLATION_TYPE_NEAREST`. La sortie est une image de profondeur transformée et une image personnalisée transformée qui doivent être allouées par l'utilisateur via l'appel de la fonction [k4a_image_create\(\)](#). Les résolutions de l'image de

profondeur transformée et de l'image personnalisée transformée doivent correspondre à la valeur `color_resolution` spécifiée lors de la création du descripteur de transformation. Par exemple, si la résolution des couleurs a été définie sur `K4A_COLOR_RESOLUTION_1080P`, la résolution de l'image de sortie doit être de 1920 x 1080 pixels. Le stride de l'image de profondeur sortie est défini sur `width * sizeof(uint16_t)`, car l'image stocke des valeurs de profondeur de 16 bits. L'image personnalisée d'entrée et l'image personnalisée transformée doivent être au format `K4A_IMAGE_FORMAT_CUSTOM8` ou `K4A_IMAGE_FORMAT_CUSTOM16`, et le stride d'image correspondant doit être défini en conséquence.

k4a_transformation_color_image_to_depth_camera

Vue d'ensemble

La fonction [k4a_transformation_color_image_to_depth_camera\(\)](#) transforme l'image en couleurs du point de vue de la caméra couleur en point de vue de la caméra de profondeur (voir la figure ci-dessus). Vous pouvez l'utiliser pour générer des images RVB-D.

Implémentation

Pour chaque pixel de la carte de profondeur, la fonction utilise la valeur de profondeur du pixel pour calculer la coordonnée de sous-pixel correspondante dans l'image en couleurs. Nous recherchons ensuite la valeur de couleur à cette coordonnée dans l'image en couleurs. Une interpolation bilinéaire est effectuée dans l'image en couleurs pour obtenir la valeur de couleur avec une précision de sous-pixel. Un pixel qui n'est pas associé à une mesure de profondeur est affecté à une valeur BGRA de `[0,0,0,0]` dans l'image de sortie. Par défaut, l'accélération GPU est activée pour cette fonction. Étant donné que cette méthode produit des trous dans l'image en couleurs transformée et ne gère pas les occlusions, nous vous recommandons d'utiliser la fonction [k4a_transformation_depth_image_to_color_camera\(\)](#) à la place.

Paramètres

Les paramètres d'entrée sont le descripteur de transformation, une image de profondeur et une image en couleurs. Les résolutions des images de profondeur et en couleurs doivent correspondre aux valeurs `depth_mode` et `color_resolution` spécifiées lors de la création du descripteur de transformation. La sortie est une image en couleurs transformée qui doit être allouée par l'utilisateur via l'appel de [k4a_image_create\(\)](#). La

résolution de l'image en couleurs transformée doit correspondre à la valeur `depth_resolution` spécifiée lors de la création du descripteur de transformation. L'image de sortie stocke quatre valeurs 8 bits représentant BGRA pour chaque pixel. Par conséquent, le stride de l'image est `width * 4 * sizeof(uint8_t)`. L'ordre des données est celui des pixels entrelacés, c'est-à-dire la valeur bleu -pixel 0, la valeur verte -pixel 0, la valeur rouge -pixel 0, la valeur alpha -pixel 0, la valeur bleue -pixel 1, et ainsi de suite.

k4a_transformation_depth_image_to_point_cloud

Vue d'ensemble

La fonction [k4a_transformation_depth_image_to_point_cloud\(\)](#) convertit une carte de profondeur 2D prise par une caméra en nuage de points 3D dans le système de coordonnées de la même caméra. La caméra peut donc être la caméra de profondeur ou la caméra couleur.

Implémentation

La fonction produit des résultats équivalents à ceux que produit la fonction [k4a_calibration_2d_to_2d\(\)](#) pour chaque pixel, mais elle est plus efficace en termes de calcul. Lors de l'appel de la fonction [k4a_transformation_create\(\)](#), nous précalculons une table de recherche-xy qui stocke les facteurs d'échelle x et y pour chaque pixel d'image. Lors de l'appel de la fonction

[k4a_transformation_depth_image_to_point_cloud\(\)](#), nous obtenons la coordonnée 3D x d'un pixel en multipliant le facteur d'échelle x du pixel par la coordonnée z du pixel. De même, la coordonnée 3D y est calculée via une multiplication avec le facteur d'échelle y. L'[exemple Fast point Cloud](#) du Kit de développement logiciel (SDK) illustre le mode de calcul de la table xy. Les utilisateurs peuvent se référer à l'exemple de code pour implémenter leur propre version de cette fonction, par exemple, pour accélérer leur pipeline GPU.

Paramètres

Les paramètres d'entrée sont le descripteur de transformation, un spécificateur de caméra et une image de profondeur. Si le spécificateur de caméra défini est la profondeur, la résolution de l'image de profondeur doit correspondre à la valeur `depth_mode` spécifiée lors de la création du descripteur de transformation. Si le spécificateur défini est la caméra couleur, la résolution doit correspondre à la valeur `color_resolution` choisie. Le paramètre de sortie est une image XYZ que l'utilisateur doit allouer via l'appel de la fonction [k4a_image_create\(\)](#). La résolution d'image XYZ doit

correspondre à celle de la carte de profondeur d'entrée. Nous stockons trois valeurs de coordonnées 16 bits signées exprimées en millimètres pour chaque pixel. Le stride d'image XYZ est donc défini sur `width * 3 * sizeof(int16_t)`. L'ordre des données est celui des pixels entrelacés, c'est-à-dire la coordonnée X -pixel 0, la coordonnée Y -pixel 0, la coordonnée Z -pixel 0, la coordonnée X -pixel 1 et ainsi de suite. Si un pixel ne peut pas être converti en 3D, la fonction affecte au pixel les valeurs `[0,0,0]`.

Exemples

[Exemple de transformation](#)

Étapes suivantes

Maintenant que vous savez comment utiliser les fonctions de transformation d'image du Kit de développement logiciel (SDK) du capteur Azure Kinect, vous pouvez également en apprendre davantage sur

[Fonctions d'étalonnage du Kit de développement logiciel \(SDK\) du capteur Kinect Azure](#)

Vous pouvez également consulter

[Systèmes de coordonnées](#)

Utiliser les fonctions d'étalonnage d'Azure Kinect

Article • 01/06/2023

Les fonctions d'étalonnage permettent de transformer des points entre les systèmes de coordonnées de chaque capteur sur l'appareil Azure Kinect. Les applications nécessitant une conversion d'images entières peuvent tirer parti des opérations accélérées disponibles dans les [fonctions de transformation](#).

Récupérer les données d'étalonnage

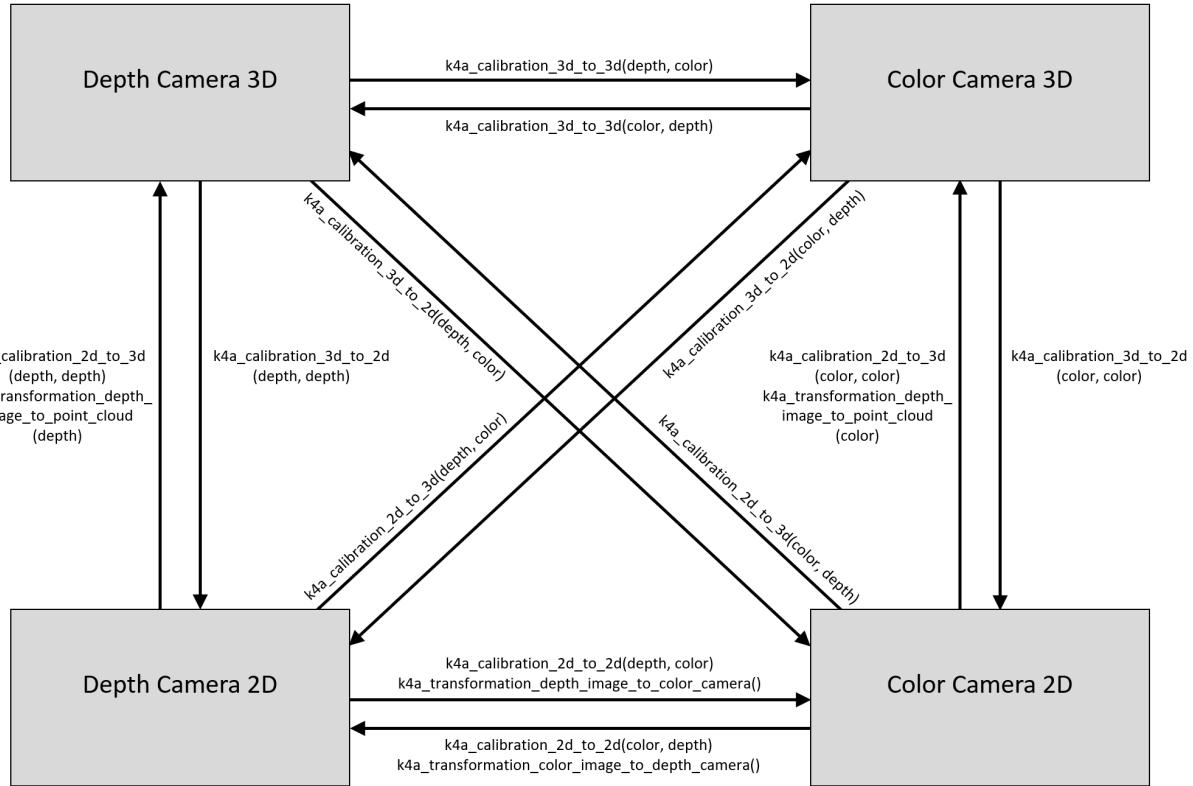
Il est nécessaire de récupérer l'étalonnage de l'appareil pour effectuer des transformations du système de coordonnées. Les données d'étalonnage sont stockées dans le type de données [k4a_calibration_t](#). Elles sont obtenue de l'appareil via la fonction [k4a_device_get_calibration\(\)](#). Les données d'étalonnage sont spécifiques non seulement de chaque appareil, mais aussi du mode de fonctionnement des caméras. Par conséquent, la fonction [k4a_device_get_calibration\(\)](#) requiert les paramètres `depth_mode` et `color_resolution` en entrée.

Compatibilité OpenCV

Les paramètres d'étalonnage sont compatibles avec la bibliothèque graphique [OpenCV](#). Pour plus d'informations sur les paramètres d'étalonnage de la caméra, consultez aussi la [documentation sur la bibliothèque graphique OpenCV](#). Consultez également l'[exemple de compatibilité OpenCV](#) du Kit de développement logiciel (SDK) qui illustre la conversion entre le type [k4a_calibration_t](#) et les structures de données OpenCV correspondantes.

Fonctions de transformation de coordonnées

La figure ci-dessous présente les différents systèmes de coordonnées d'Azure Kinect, ainsi que les fonctions permettant de les convertir entre eux. Pour que la figure reste simple, nous n'indiquons pas les systèmes de coordonnées 3D de gyroscope et d'accéléromètre.



Remarque concernant la distorsion de l'objectif : les coordonnées 2D font toujours référence à l'image distordue dans le Kit de développement logiciel (SDK). L'[exemple de correction de distorsion](#) du Kit de développement logiciel (SDK) montre la correction de distorsion d'image. En général, les points 3D ne sont jamais affectés par la distorsion de l'objectif.

Opérer une conversion entre systèmes de coordonnées 3D

La fonction [k4a_calibration_3d_to_3d\(\)](#) convertit un point 3D du système de coordonnées source en point 3D du système de coordonnées cible en utilisant l'étalonnage extrinsèque de la caméra. La source et la cible peuvent être définies sur l'un des quatre systèmes de coordonnées 3D, à savoir, caméra couleur, caméra de profondeur, gyroscope ou accéléromètre. Si la source et la cible sont identiques, le point 3D d'entrée non modifié est retourné en tant que sortie.

Conversion entre systèmes de coordonnées 2D et 3D

La fonction [k4a_calibration_3d_to_2d\(\)](#) convertit un point 3D du système de coordonnées source en coordonnée de pixel 2D de la caméra cible. Cette fonction est souvent appelée fonction de projection. Si la source peut être définie sur l'un des quatre systèmes de coordonnées 3D, la cible doit être la caméra de profondeur ou couleur. Si la source et la cible sont différentes, le point d'entrée 3D est converti dans le système de coordonnées 3D de la caméra cible à l'aide de la fonction [k4a_calibration_3d_to_3d\(\)](#).

Une fois le point 3D représenté dans le système de coordonnées de la caméra cible, les coordonnées de pixel 2D correspondantes sont calculées en utilisant l'étalonnage intrinsèque de la caméra cible. Si un point 3D se trouve en dehors de la zone visible de la caméra cible, la valeur valide est définie sur 0.

La fonction [k4a_calibration_2d_to_3d\(\)](#) convertit une coordonnée de pixel 2D de la caméra source en un point 3D du système de coordonnées de la caméra cible. La source doit être une caméra couleur ou de profondeur. La cible peut être définie sur l'un des quatre systèmes de coordonnées 3D. En plus de la coordonnée de pixel 2D, la valeur de profondeur du pixel (en millimètres) dans l'image de la caméra source est requise comme entrée pour la fonction. Une façon de dériver la valeur de profondeur dans la géométrie de la caméra couleur consiste à utiliser la fonction [k4a_transformation_depth_image_to_color_camera\(\)](#). La fonction calcule le rayon 3D qui mène du point focal de la caméra source à la coordonnée de pixel spécifiée en utilisant l'étalonnage intrinsèque de la caméra source. La valeur de profondeur est ensuite utilisée pour rechercher l'emplacement exact du point 3D sur ce rayon. Cette opération est souvent appelée fonction de déprojection. Si les caméras source et cible sont différentes, la fonction transforme le point 3D au système de coordonnées de la cible via la fonction [k4a_calibration_3d_to_3d\(\)](#). Si une coordonnée de pixel 2D se trouve en dehors de la zone visible de la caméra source, la valeur valide est définie sur 0.

Conversion entre systèmes de coordonnées 2D

La fonction [k4a_calibration_2d_to_2d\(\)](#) convertit une coordonnée de pixel 2D de la caméra source en coordonnée de pixel 2D de la caméra cible. La source et la cible définies doivent être une caméra couleur ou de profondeur. La fonction requiert la valeur de profondeur du pixel (en millimètres) dans l'image de la caméra source en entrée. Une façon de dériver la valeur de profondeur dans la géométrie de la caméra de couleur consiste à utiliser la fonction

[k4a_transformation_depth_image_to_color_camera\(\)](#). Elle appelle la fonction [k4a_calibration_2d_to_3d\(\)](#) pour effectuer la conversion en un point 3D du système de la caméra source. Elle appelle ensuite la fonction [k4a_calibration_3d_to_2d\(\)](#) pour convertir en coordonnée de pixel 2D de l'image de la caméra cible. La valeur valide est définie sur 0, si la fonction [k4a_calibration_2d_to_3d\(\)](#) ou [k4a_calibration_3d_to_2d\(\)](#) retourne un résultat non valide.

Exemples connexes

- [Exemple de compatibilité OpenCV](#)
- [Exemple de correction de distorsion](#)

- Exemple de nuage de points rapide ↗

Étapes suivantes

Maintenant que vous savez ce que sont les étalonnages de caméra, vous pouvez également apprendre à

[Capturer la synchronisation des appareils](#)

Vous pouvez également consulter

[Systèmes de coordonnées](#)

Capturer la synchronisation d'appareils Azure Kinect

Article • 01/06/2023

Le matériel Azure Kinect permet d'aligner le temps de capture des images de couleur et de profondeur. L'alignement entre les caméras d'un même appareil est une **synchronisation interne**. L'alignement du temps de capture sur plusieurs appareils connectés est une **synchronisation externe**. Le réseau de microphones fonctionne indépendamment des caméras de profondeur et couleur.

Synchronisation interne de l'appareil

La capture d'images entre les différentes caméras est synchronisée sur le matériel. Dans chaque `k4a_capture_t` qui contient des images provenant du capteur de couleur et de profondeur, les horodatages des images sont alignés en fonction du mode de fonctionnement du matériel. Par défaut, les images d'une capture sont centrées sur l'exposition. La chronologie relative des captures de profondeur et de couleur peut être ajustée à l'aide du champ `depth_delay_off_color_usec` de `k4a_device_configuration_t`.

Synchronisation externe de l'appareil

Consultez [Configurer la synchronisation externe](#) pour en savoir plus sur la configuration matérielle.

Le logiciel de chaque appareil connecté doit être configuré pour fonctionner en mode **principal** ou **subordonné**. Ce paramètre est configuré dans la structure `k4a_device_configuration_t`.

Lorsque vous utilisez la synchronisation externe, les caméras subordonnées doivent toujours être démarrées avant la caméra principale pour que les horodatages s'alignent correctement.

Mode subordonné

C

```
k4a_device_configuration_t deviceConfig;
deviceConfig.wired_sync_mode = K4A_WIRED_SYNC_MODE_SUBORDINATE
```

Mode principal

C

```
k4a_device_configuration_t deviceConfig;  
deviceConfig.wired_sync_mode = K4A_WIRED_SYNC_MODE_MASTER;
```

Récupération de l'état des prises de synchronisation

Pour récupérer par programmation l'état actuel des prises d'entrée et de sortie de synchronisation, utilisez la fonction [k4a_device_get_sync_jack](#).

Étapes suivantes

Vous savez maintenant comment activer et capturer la synchronisation des appareils.

Vous pouvez également consulter la rubrique suivante :

[API d'enregistrement et de lecture du Kit de développement logiciel \(SDK\) Sensor Azure Kinect](#)

API de lecture Azure Kinect

Article • 01/06/2023

Le Kit de développement logiciel (SDK) de capteur fournit une API pour l'enregistrement des données de l'appareil dans un fichier Matroska (.mkv). Le format de conteneur Matroska stocke les pistes vidéo, les échantillons d'IMU et l'étalonnage des appareils. Vous pouvez générer des enregistrements à l'aide de l'utilitaire de ligne de commande [k4arecorder](#) fourni. Vous pouvez également personnaliser et enregistrer des enregistrements directement à l'aide de l'API d'enregistrement.

Pour plus d'informations sur l'API d'enregistrement, consultez [k4a_record_create\(\)](#).

Pour plus d'informations sur les spécifications de format de fichier Matroska, consultez la page [Format de fichier d'enregistrement](#).

Utiliser l'API de lecture

Vous pouvez ouvrir des fichiers d'enregistrement à l'aide de l'API de lecture. L'API de lecture donne accès aux données de capteur dans le même format que le reste du Kit de développement logiciel (SDK) du capteur.

Ouvrir un fichier d'enregistrement

Dans l'exemple suivant, nous ouvrons un enregistrement à l'aide de la commande [k4a_playback_open\(\)](#), imprimons la longueur de l'enregistrement, puis fermons le fichier à l'aide de la commande [k4a_playback_close\(\)](#).

C

```
k4a_playback_t playback_handle = NULL;
if (k4a_playback_open("recording.mkv", &playback_handle) != K4A_RESULT_SUCCEEDED)
{
    printf("Failed to open recording\n");
    return 1;
}

uint64_t recording_length = k4a_playback_get_last_timestamp_usec(playback_handle);
printf("Recording is %lld seconds long\n", recording_length / 1000000);

k4a_playback_close(playback_handle);
```

Lire les captures

Une fois le fichier ouvert, nous pouvons commencer à lire les captures de l'enregistrement. L'exemple suivant lit chacune des captures dans le fichier.

C

```
k4a_capture_t capture = NULL;
k4a_stream_result_t result = K4A_STREAM_RESULT_SUCCEEDED;
while (result == K4A_STREAM_RESULT_SUCCEEDED)
```

```

{
    result = k4a_playback_get_next_capture(playback_handle, &capture);
    if (result == K4A_STREAM_RESULT_SUCCEEDED)
    {
        // Process capture here
        k4a_capture_release(capture);
    }
    else if (result == K4A_STREAM_RESULT_EOF)
    {
        // End of file reached
        break;
    }
}
if (result == K4A_STREAM_RESULT_FAILED)
{
    printf("Failed to read entire recording\n");
    return 1;
}

```

Rechercher dans un enregistrement

Une fois que nous avons atteint la fin du fichier, nous pouvons revenir en arrière et le relire. Vous pouvez effectuer ce processus en lisant en arrière à l'aide de la commande [k4a_playback_get_previous_capture\(\)](#), mais cela peut prendre beaucoup de temps selon la durée de l'enregistrement. Au lieu de cela, nous pouvons utiliser la fonction [k4a_playback_seek_timestamp\(\)](#) pour atteindre un point spécifique dans le fichier.

Dans cet exemple, nous spécifions des horodateurs en microsecondes pour rechercher plusieurs points dans le fichier.

C

```

// Seek to the beginning of the file
if (k4a_playback_seek_timestamp(playback_handle, 0, K4A_PLAYBACK_SEEK_BEGIN) != K4A_RESULT_SUCCEEDED)
{
    return 1;
}

// Seek to the end of the file
if (k4a_playback_seek_timestamp(playback_handle, 0, K4A_PLAYBACK_SEEK_END) != K4A_RESULT_SUCCEEDED)
{
    return 1;
}

// Seek to 10 seconds from the start
if (k4a_playback_seek_timestamp(playback_handle, 10 * 1000000, K4A_PLAYBACK_SEEK_BEGIN) != K4A_RESULT_SUCCEEDED)
{
    return 1;
}

// Seek to 10 seconds from the end
if (k4a_playback_seek_timestamp(playback_handle, -10 * 1000000, K4A_PLAYBACK_SEEK_END) != K4A_RESULT_SUCCEEDED)
{

```

```
    return 1;  
}
```

Lire les Informations sur les balises

Les enregistrements peuvent également contenir diverses métadonnées, telles que le numéro de série et les versions du microprogramme de l'appareil. Ces métadonnées sont stockées dans des balises d'enregistrement, accessibles à l'aide de la fonction [k4a_playback_get_tag\(\)](#).

C

```
// Print the serial number of the device used to record  
char serial_number[256];  
size_t serial_number_size = 256;  
k4a_buffer_result_t buffer_result = k4a_playback_get_tag(playback_handle,  
"K4A_DEVICE_SERIAL_NUMBER", &serial_number, &serial_number_size);  
if (buffer_result == K4A_BUFFER_RESULT_SUCCEEDED)  
{  
    printf("Device serial number: %s\n", serial_number);  
}  
else if (buffer_result == K4A_BUFFER_RESULT_TOO_SMALL)  
{  
    printf("Device serial number too long.\n");  
}  
else  
{  
    printf("Tag does not exist. Device serial number was not recorded.\n");  
}
```

Liste de balises d'enregistrement

Vous trouverez ci-dessous la liste de toutes les balises par défaut qui peuvent être incluses dans un fichier d'enregistrement. La plupart de ces valeurs sont disponibles dans le cadre de la structure [k4a_record_configuration_t](#), et peuvent être lues avec la fonction [k4a_playback_get_record_configuration\(\)](#).

Si une balise n'existe pas, elle est supposée avoir la valeur par défaut.

| Nom de la balise | Valeur par défaut | k4a_record_configuration_t Champ | Notes |
|------------------|-------------------|---|---|
| K4A_COLOR_MODE | "OFF" | color_format / color_resolution | Valeurs possibles : "OFF", "MJPG_1080P", "NV12_720P", "YUY2_720P", etc. |
| K4A_DEPTH_MODE | "OFF" | depth_mode / depth_track_enabled | Valeurs possibles : "OFF", "NFOV_UNBINNED", "PASSIVE_IR", etc. |
| K4A_IR_MODE | "OFF" | depth_mode / ir_track_enabled | Valeurs possibles : "OFF", "ACTIVE", "PASSIVE" |
| K4A_IMU_MODE | "OFF" | imu_track_enabled | Valeurs possibles : "ON", "OFF" |

| Nom de la balise | Valeur par défaut | <code>k4a_record_configuration_t</code> Champ | Notes |
|---|--------------------|--|---|
| <code>K4A_CALIBRATION_FILE</code> | "calibration.json" | N/A | Consultez k4a_device_get_raw_calibration() |
| <code>K4A_DEPTH_DELAY_NS</code> | "0" | <code>depth_delay_off_color_usec</code> | Valeur stockée en nanosecondes, l'API fournit des microsecondes. |
| <code>K4A_WIRED_SYNC_MODE</code> | "STANDALONE" | <code>wired_sync_mode</code> | Valeurs possibles : "STANDALONE", "MASTER", "SUBORDINATE" |
| <code>K4A_SUBORDINATE_DELAY_NS</code> | "0" | <code>subordinate_delay_off_master_usec</code> | Valeur stockée en nanosecondes, l'API fournit des microsecondes. |
| <code>K4A_COLOR_FIRMWARE_VERSION</code> | "" | N/A | Version du microprogramme de couleur de l'appareil, par exemple « 1.x.xx » |
| <code>K4A_DEPTH_FIRMWARE_VERSION</code> | "" | N/A | Version du microprogramme de profondeur de l'appareil, par exemple « 1.x.xx » |
| <code>K4A_DEVICE_SERIAL_NUMBER</code> | "" | N/A | Numéro de série de l'appareil d'enregistrement |
| <code>K4A_START_OFFSET_NS</code> | "0" | <code>start_timestamp_offset_usecs</code> | Consultez Synchronisation d'horodatage ci-dessous. |
| <code>K4A_COLOR_TRACK</code> | None | N/A | Consultez Format de fichier d'enregistrement – Identification de pistes . |
| <code>K4A_DEPTH_TRACK</code> | None | N/A | Consultez Format de fichier d'enregistrement – Identification de pistes . |
| <code>K4A_IR_TRACK</code> | None | N/A | Consultez Format de fichier d'enregistrement – Identification de pistes . |
| <code>K4A_IMU_TRACK</code> | None | N/A | Consultez Format de fichier d'enregistrement – Identification de pistes . |

Synchronisation d'horodatage

Le format Matroska nécessite que les enregistrements commencent par un horodateur à zéro. Lors de la [synchronisation externe des caméras](#), le premier horodatage de chaque appareil peut être différent de zéro.

Pour conserver les horodatages d'origine des appareils entre l'enregistrement et la lecture, le fichier stocke un décalage à appliquer aux horodatages.

La balise `K4A_START_OFFSET_NS` permet de spécifier un décalage d'horodatage afin que les fichiers puissent être resynchronisés après l'enregistrement. Vous pouvez ajouter ce décalage d'horodatage à chaque horodatage dans le fichier pour reconstruire les horodatages de l'appareil d'origine.

Le décalage de début est également disponible dans la structure [k4a_record_configuration_t](#).

Obtenir les résultats Body Tracking

Article • 24/05/2023

Le Kit de développement logiciel (SDK) de suivi de corps utilise un objet dédié au suivi de corps pour traiter les captures Azure Kinect DK, et génère des résultats de suivi de corps. Il gère également l'état global du dispositif de suivi, le traitement des files d'attente et la file d'attente de sortie. L'utilisation du dispositif de suivi de corps comporte trois étapes :

- Créer un dispositif de suivi
- Capturer des images de profondeur et IR à l'aide d'un appareil Azure Kinect DK
- Mettre en file d'attente la capture et afficher les résultats

Créer un dispositif de suivi

La première étape de l'utilisation du suivi de corps consiste à créer un dispositif de suivi et à suivre la procédure d'étalonnage du capteur [k4a_calibration_t](#). L'étalonnage du capteur peut être interrogé à l'aide de la fonction [k4a_device_get_calibration\(\)](#) du Kit de développement logiciel (SDK) du capteur Azure Kinect.

C

```
k4a_calibration_t sensor_calibration;
if (K4A_RESULT_SUCCEEDED != k4a_device_get_calibration(device,
device_config.depth_mode, K4A_COLOR_RESOLUTION_OFF, &sensor_calibration))
{
    printf("Get depth camera calibration failed!\n");
    return 0;
}

k4abt_tracker_t tracker = NULL;
k4abt_tracker_configuration_t tracker_config = K4ABT_TRACKER_CONFIG_DEFAULT;
if (K4A_RESULT_SUCCEEDED != k4abt_tracker_create(&sensor_calibration,
tracker_config, &tracker))
{
    printf("Body tracker initialization failed!\n");
    return 0;
}
```

Capturer des images de profondeur et IR

La capture d'images à l'aide d'Azure Kinect DK est traitée dans la page [Récupérer les images](#).

ⓘ Notes

Les modes `K4A_DEPTH_MODE_NFOV_UNBINNED` ou `K4A_DEPTH_MODE_WFOV_2X2BINNED` sont recommandés pour bénéficier de performances et d'une précision optimales. N'utilisez pas les modes `K4A_DEPTH_MODE_OFF` ou `K4A_DEPTH_MODE_PASSIVE_IR`.

Les modes d'Azure Kinect DK pris en charge sont décrits dans les [spécifications matérielles](#) d'Azure Kinect DK et les énumérations `k4a_depth_mode_t`.

C

```
// Capture a depth frame
switch (k4a_device_get_capture(device, &capture, TIMEOUT_IN_MS))
{
    case K4A_WAIT_RESULT_SUCCEEDED:
        break;
    case K4A_WAIT_RESULT_TIMEOUT:
        printf("Timed out waiting for a capture\n");
        continue;
        break;
    case K4A_WAIT_RESULT_FAILED:
        printf("Failed to read a capture\n");
        goto Exit;
}
```

Empiler la capture et dépiler les résultats

Le dispositif de suivi gère en interne une file d'attente d'entrée et une file d'attente de sortie pour traiter de manière asynchrone et plus efficace les captures d'Azure Kinect DK. Utilisez la fonction [k4abt_tracker_enqueue_capture\(\)](#) pour ajouter une nouvelle capture à la file d'attente d'entrée. Utilisez la fonction [k4abt_tracker_pop_result\(\)](#) pour afficher un résultat de la file d'attente de sortie. L'utilisation de la valeur de délai d'attente dépend de l'application et contrôle la valeur de délai d'attente de la file d'attente.

Traitements sans attente

Utilisez ce modèle pour les applications monothread qui ont besoin de résultats immédiats et peuvent prendre en charge les trames supprimées (par exemple l'affichage de vidéos en direct à partir d'un appareil). L'exemple `simple_3d_viewer` situé dans [GitHub Azure-Kinect-Samples](#) constitue un exemple de traitement sans attente.

C

```

k4a_wait_result_t queue_capture_result =
k4abt_tracker_enqueue_capture(tracker, sensor_capture, 0);
k4a_capture_release(sensor_capture); // Remember to release the sensor
capture once you finish using it
if (queue_capture_result == K4A_WAIT_RESULT_FAILED)
{
    printf("Error! Adding capture to tracker process queue failed!\n");
    break;
}

k4abt_frame_t body_frame = NULL;
k4a_wait_result_t pop_frame_result = k4abt_tracker_pop_result(tracker,
&body_frame, 0);
if (pop_frame_result == K4A_WAIT_RESULT_SUCCEEDED)
{
    // Successfully popped the body tracking result. Start your processing
    ...

    k4abt_frame_release(body_frame); // Remember to release the body frame
once you finish using it
}

```

Traitement avec attente

Utilisez ce modèle pour les applications qui n'ont pas besoin de résultats pour chaque trame (par exemple le traitement d'une vidéo à partir d'un fichier). L'exemple `simple_sample.exe` situé dans [GitHub Azure-Kinect-Samples](#) constitue un exemple de traitement avec attente.

```

C

k4a_wait_result_t queue_capture_result =
k4abt_tracker_enqueue_capture(tracker, sensor_capture, K4A_WAIT_INFINITE);
k4a_capture_release(sensor_capture); // Remember to release the sensor
capture once you finish using it
if (queue_capture_result != K4A_WAIT_RESULT_SUCCEEDED)
{
    // It should never hit timeout or error when K4A_WAIT_INFINITE is set.
    printf("Error! Adding capture to tracker process queue failed!\n");
    break;
}

k4abt_frame_t body_frame = NULL;
k4a_wait_result_t pop_frame_result = k4abt_tracker_pop_result(tracker,
&body_frame, K4A_WAIT_INFINITE);
if (pop_frame_result != K4A_WAIT_RESULT_SUCCEEDED)
{
    // It should never hit timeout or error when K4A_WAIT_INFINITE is set.
    printf("Error! Popping body tracking result failed!\n");
    break;
}

```

```
}
```

```
// Successfully popped the body tracking result. Start your processing
```

```
...
```

```
k4abt_frame_release(body_frame); // Remember to release the body frame once
```

```
you finish using it
```

Étapes suivantes

[Accéder aux données de l'ossature du corps](#)

Accéder aux données de l'ossature du corps

Article • 01/06/2023

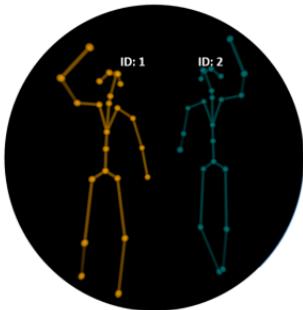
Cet article décrit les données contenues dans une image de corps et les fonctions permettant d'accéder à celles-ci.

Voici les fonctions qui sont abordées ici :

- [k4abt_frame_get_body_id\(\)](#)
- [k4abt_frame_get_body_index_map\(\)](#)
- [k4abt_frame_get_body_skeleton\(\)](#)
- [k4abt_frame_get_capture\(\)](#)
- [k4abt_frame_get_num_bodies\(\)](#)
- [k4abt_frame_get_device_timestamp_usec\(\)](#)

Composants clés d'une image de corps

Chaque image de corps se compose d'une collection de structures de corps, d'un mappage d'index de corps 2D et de la capture d'entrée ayant produit ce résultat.



A collection of body structs

Each body struct contains:
• Body ID – the tracked ID of the body.
• Joint Positions – the 3D position of each joint.
• Joint Orientations – the orientation of each joint coordinate frame represented in quaternion.



2D body index map

The 2D instance segmentation map segments each body instance from the background.



Input capture

The input capture that used to generate the body tracking results.

Accéder à la collection de structures de corps

Plusieurs corps peuvent être détectés dans une seule capture. Vous pouvez interroger le nombre de corps en appelant la fonction [k4abt_frame_get_num_bodies\(\)](#).

C

```
size_t num_bodies = k4abt_frame_get_num_bodies(body_frame);
```

Vous utilisez les fonctions [k4abt_frame_get_body_id\(\)](#) et [k4abt_frame_get_body_skeleton\(\)](#) pour opérer une itération dans chaque index de corps afin de rechercher l'ID de corps et les informations de position/orientation jointes.

```
C

for (size_t i = 0; i < num_bodies; i++)
{
    k4abt_skeleton_t skeleton;
    k4abt_frame_get_body_skeleton(body_frame, i, &skeleton);
    uint32_t id = k4abt_frame_get_body_id(body_frame, i);
}
```

Accéder au mappage d'index de corps

Vous utilisez la fonction [k4abt_frame_get_body_index_map\(\)](#) pour accéder au mappage d'index de corps. Pour une explication détaillée du mappage d'index de corps, consultez [Mappage d'index de corps](#). Veillez à libérer le mappage d'index de corps quand il n'est plus nécessaire.

```
C

k4a_image_t body_index_map = k4abt_frame_get_body_index_map(body_frame);
... // Do your work with the body index map
k4a_image_release(body_index_map);
```

Accéder à la capture d'entrée

Le dispositif de suivi du corps est une API asynchrone. La capture d'origine a peut-être déjà été libérée au moment où le résultat s'affiche. Utilisez la fonction [k4abt_frame_get_capture\(\)](#) pour interroger la capture d'entrée utilisée pour générer ce résultat de suivi du corps. Le nombre de références pour k4a_capture_t est augmenté à chaque appel de cette fonction. Utilisez la fonction [k4a_capture_release\(\)](#) lorsque la capture n'est plus nécessaire.

```
C

k4a_capture_t input_capture = k4abt_frame_get_capture(body_frame);
... // Do your work with the input capture
k4a_capture_release(input_capture);
```

Étapes suivantes

[Kit de développement logiciel \(SDK\) de suivi du corps d'Azure Kinect](#)

Ajouter une bibliothèque Azure Kinect à votre projet Visual Studio

Article • 01/06/2023

Cet article vous guide tout au long du processus d'ajout d'un package NuGet Azure Kinect à votre projet Visual Studio.

Installer le package NuGet Azure Kinect

Pour installer le package NuGet Azure Kinect :

1. Vous trouverez des instructions détaillées sur l'installation d'un package NuGet dans Visual Studio dans [Démarrage rapide : Installer et utiliser un package dans Visual Studio](#).
2. Pour ajouter le package, vous pouvez utiliser l'interface utilisateur du gestionnaire de package en cliquant avec le bouton droit sur références et en sélectionnant Gérer les packages NuGet dans l'Explorateur de solutions.
3. Choisissez [nuget.org ↗](#) comme source du package, sélectionnez l'onglet Parcourir, puis recherchez `Microsoft.Azure.Kinect.Sensor`.
4. Sélectionnez ce package dans la liste, puis installez-le.

Utiliser le package NuGet Azure Kinect

Une fois le package ajouté, ajoutez le fichier d'en-tête inclus au code source, par exemple :

- `#include <k4a/k4a.h>`
- `#include <k4arecord/record.h>`
- `#include <k4arecord/playback.h>`

Étapes suivantes

You are now ready to create your first application

Mettre à jour le microprogramme Azure Kinect DK

Article • 01/06/2023

Ce document fournit des conseils sur la façon de mettre à jour le microprogramme de votre appareil Azure Kinect DK.

Azure Kinect DK ne met pas à jour le microprogramme automatiquement. Vous pouvez utiliser l'[outil du microprogramme Azure Kinect](#) pour mettre à jour le microprogramme manuellement vers la dernière version disponible.

Préparer la mise à jour du microprogramme

1. [Télécharger le SDK](#).
2. Installez le kit SDK.
3. Dans l'emplacement d'installation du Kit de développement logiciel (SDK) sous (emplacement d'installation du SDK)\tools\, vous devriez trouver :
 - AzureKinectFirmwareTool.exe
 - Fichier .bin de microprogramme dans le dossier du microprogramme, par exemple *AzureKinectDK_Fw_1.5.926614.bin*.
4. Connectez votre appareil à l'ordinateur hôte et mettez-le en marche.

Important

Veillez à ce que les connexions USB et d'alimentation soient correctement établies pendant la mise à jour du microprogramme. La suppression de l'une ou l'autre risque d'altérer le microprogramme.

Mettre à jour le microprogramme de l'appareil

1. Ouvrez une invite de commandes dans le dossier (emplacement d'installation du SDK)\tools\.
2. Mettre à jour le microprogramme à l'aide de l'outil de microprogramme Azure Kinect

```
AzureKinectFirmwareTool.exe -u <device_firmware_file.bin>
```

Exemple :

```
AzureKinectFirmwareTool.exe -u firmware\AzureKinectDK_Fw_1.5.926614.bin
```

3. Attendez la fin de la mise à jour du microprogramme. Cela peut prendre quelques minutes en fonction de la taille de l'image.

Vérifier la version du microprogramme de l'appareil

1. Vérifiez que le microprogramme est à jour.

```
AzureKinectFirmwareTool.exe -q
```

2. Consultez l'exemple suivant.

```
Console

>AzureKinectFirmwareTool.exe -q
== Azure Kinect DK Firmware Tool ==
Device Serial Number: 000805192412
Current Firmware Versions:
RGB camera firmware:      1.6.102
Depth camera firmware:    1.6.75
Depth config file:        6109.7
Audio firmware:            1.6.14
Build Config:              Production
Certificate Type:          Microsoft
```

3. Si vous voyez la sortie ci-dessus, cela signifie que le microprogramme est à jour.
4. Après la mise à jour du microprogramme, vous pouvez exécuter la [visionneuse Azure Kinect](#) pour vérifier que tous les capteurs fonctionnent comme prévu.

Dépannage

Les mises à jour du microprogramme peuvent échouer pour plusieurs raisons. En cas d'échec d'une mise à jour de microprogramme, essayez de suivre les étapes d'atténuation suivantes :

1. Réessayez d'exécuter la commande de mise à jour du microprogramme.
2. Vérifiez que l'appareil est toujours connecté en interrogeant la version du microprogramme. AzureKinectFirmwareTool.exe

3. Si tout échoue, suivez les étapes de [réécupération](#) pour revenir au microprogramme d'usine, puis réessayez.

Pour tout problème supplémentaire, consultez les [pages de support de Microsoft](#)

Étapes suivantes

[Outil du microprogramme Azure Kinect](#)

Utiliser l'enregistreur Kinect Azure avec des appareils externes synchronisés

Article • 01/06/2023

Cet article fournit des conseils sur la façon dont l'[Enregistreur Azure Kinect](#) peut enregistrer des données d'appareils configurés pour une synchronisation externe.

Prérequis

- [Configurez plusieurs unités Azure Kinect DK pour la synchronisation externe ↗](#).

Contraintes de synchronisation externe

- L'appareil maître ne peut pas avoir de câble SYNC IN connecté.
- L'appareil maître doit diffuser la caméra RVB pour permettre la synchronisation.
- Toutes les unités doivent utiliser la même configuration de caméra (taux de trames et résolution).
- Toutes les unités doivent exécuter le même microprogramme d'appareil ([instruction de mise à jour du microprogramme](#)).
- Tous les appareils subordonnés doivent être démarrés avant l'appareil maître.
- La même valeur d'exposition doit être définie sur tous les appareils.
- Le paramètre de *retard par rapport au maître* de chaque appareil subordonné est relatif à l'appareil maître.

Enregistrer lorsque chaque unité a un PC hôte

Dans l'exemple ci-dessous, chaque appareil a son propre PC hôte dédié. Il est recommandé de connecter les appareils à des PC dédiés afin d'éviter les problèmes liés à l'utilisation de la bande passante USB et de l'UC/GPU.

Subordonné-1

1. Configurez un enregistreur pour la première unité

```
k4arecorder.exe --external-sync sub -e -8 -r 5 -l 10 sub1.mkv
```

2. L'appareil commence à attendre

Console

```
Device serial number: 000011590212
Device version: Rel; C: 1.5.78; D: 1.5.60[6109.6109]; A: 1.5.13
Device started
[subordinate mode] Waiting for signal from master
```

Subordonné-2

1. Configurez un enregistreur pour la deuxième unité

```
k4arecorder.exe --external-sync sub -e -8 -r 5 -l 10 sub2.mkv
```

2. L'appareil commence à attendre

```
Console

Device serial number: 000011590212
Device version: Rel; C: 1.5.78; D: 1.5.60[6109.6109]; A: 1.5.13
Device started
[subordinate mode] Waiting for signal from master
```

Master

1. Démarrez l'enregistrement sur le maître

```
>k4arecorder.exe --external-sync master -e -8 -r 5 -l 10 master.mkv
```

2. Attendez la fin de l'enregistrement

Enregistrement lorsque plusieurs unités sont connectées à un seul PC hôte

Plusieurs Azure Kinect DK peuvent être connectés à un seul PC hôte. Toutefois, cela peut être très exigeant pour la bande passante USB et le calcul de l'hôte. Pour réduire la demande :

- Connectez chaque appareil à son propre contrôleur d'hôte USB.
- Dotez-vous d'un CPU puissant capable de gérer le moteur de profondeur pour chaque appareil.
- Enregistrez uniquement les capteurs nécessaires et utilisez un taux de trames inférieur.

Commencez toujours par démarrer les appareils subordonnés avant l'appareil maître.

Subordonné-1

1. Démarrez l'enregistreur sur l'appareil subordonné

```
>k4arecorder.exe --device 1 --external-sync subordinate --imu OFF -e -8 -r 5 -  
1 5 output-2.mkv
```

2. L'appareil passe en état d'attente

Master

1. Démarrez l'appareil maître

```
>k4arecorder.exe --device 0 --external-sync master --imu OFF -e -8 -r 5 -l 5  
output-1.mkv
```

2. Attendez la fin de l'enregistrement

Lecture de l'enregistrement

Vous pouvez utiliser la [Visionneuse Azure Kinect](#) pour lire un enregistrement.

Conseils

- Utilisez l'exposition manuelle pour l'enregistrement de caméras synchronisées. L'exposition automatique de la caméra RVB peut avoir une incidence sur la synchronisation.
- Un redémarrage de l'appareil subordonné entraîne une perte de la synchronisation.
- Certains [modes de l'appareil photo](#) prennent en charge au maximum 15 i/s. Nous vous recommandons de ne pas mélanger les modes/fréquences d'images entre les appareils
- La connexion de plusieurs unités à un seul PC peut facilement saturer la bande passante USB. Envisagez d'utiliser un PC hôte distinct par appareil. Faites également attention au calcul de l'UC/GPU.
- Désactivez le microphone et les IMU s'ils ne sont pas nécessaires pour améliorer la fiabilité.

Pour tout problème, consultez [Résolution des problèmes](#)

Voir aussi

- [Configurer une synchronisation externe ↗](#)
- [Enregistreur Azure Kinect](#) pour les paramètres de l'enregistreur et des informations supplémentaires.
- [Visionneuse Azure Kinect](#) pour la lecture des enregistrements ou la définition des propriétés de caméra RVB non disponible via l'enregistreur.
- [Outil du microprogramme Kinect Azure](#) pour la mise à jour du microprogramme de l'appareil.

Visionneuse Azure Kinect

Article • 01/06/2023

La visionneuse Azure Kinect qui se trouve dans le répertoire des outils installés comme `k4aviewer.exe` (par exemple, `C:\Program Files\Azure Kinect SDK vX.Y.Z\tools\k4aviewer.exe`, où `X.Y.Z` est la version installée du Kit de développement logiciel (SDK)), permet de visualiser tous les flux de données d'appareils aux fins suivantes :

- Vérifier que les capteurs fonctionnent correctement.
- Positionner l'appareil.
- Expérimenter des paramètres de caméra.
- Lire une configuration d'appareil.
- Lire des enregistrements effectués avec l'[Enregistreur Azure Kinect](#).

Pour plus d'informations sur la Visionneuse Azure Kinect, regardez la [vidéo Comment utiliser Azure Kinect](#).

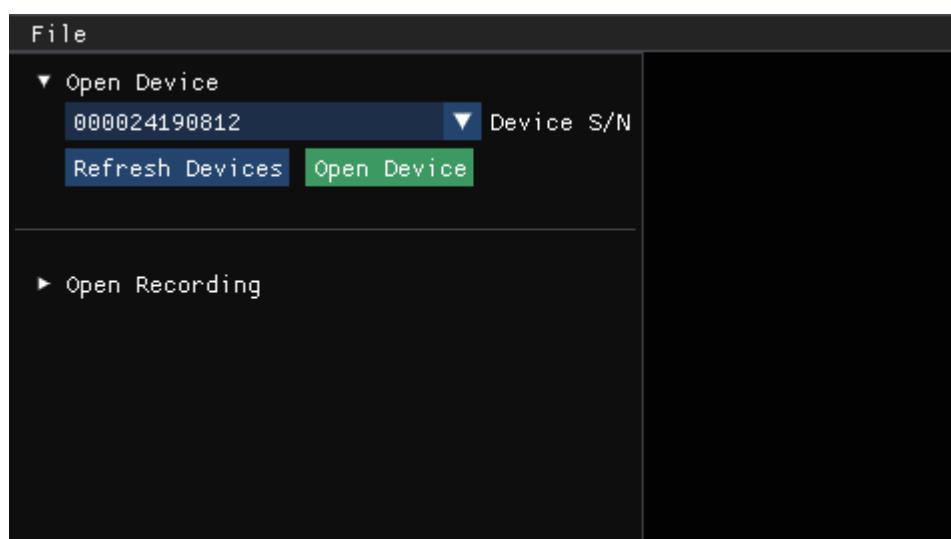
La visionneuse Azure Kinect est disponible en [open source](#) et peut être utilisée en tant qu'exemple pour l'utilisation des API.

Utiliser la visionneuse

La visionneuse peut fonctionner en deux modes : avec des données actives du capteur ou à partir de données enregistrées ([Enregistreur Azure Kinect](#)).

Démarrer l'application

Lancez l'application en exécutant `k4aviewer.exe`.



Utiliser la visionneuse avec des données actives

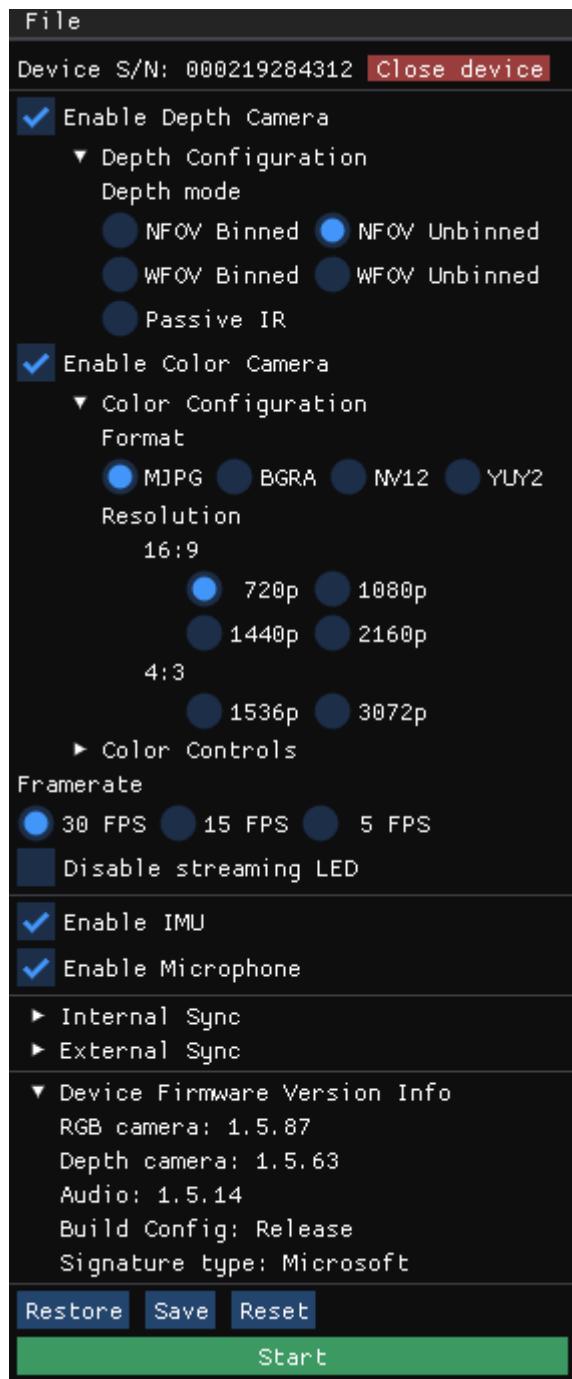
1. Dans la section **Open Device** (Ouvrir un appareil), sélectionnez le **Serial Number** (Numéro de série) de l'appareil à ouvrir. Ensuite, sélectionnez **Refresh** (Actualiser) si l'appareil est manquant.
2. Sélectionnez le bouton **Open Device** (Ouvrir un appareil).
3. Sélectionnez **Start** (Démarrer) pour commencer la diffusion en continu des données avec les paramètres par défaut.

Utiliser la visionneuse avec des données enregistrées

Dans la section **Open Recording** (Ouvrir un enregistrement), accédez au fichier enregistré et sélectionnez-le.

Vérifier la version du microprogramme de l'appareil

Accédez à la version du microprogramme de l'appareil dans la fenêtre de configuration, comme illustré dans l'image suivante.



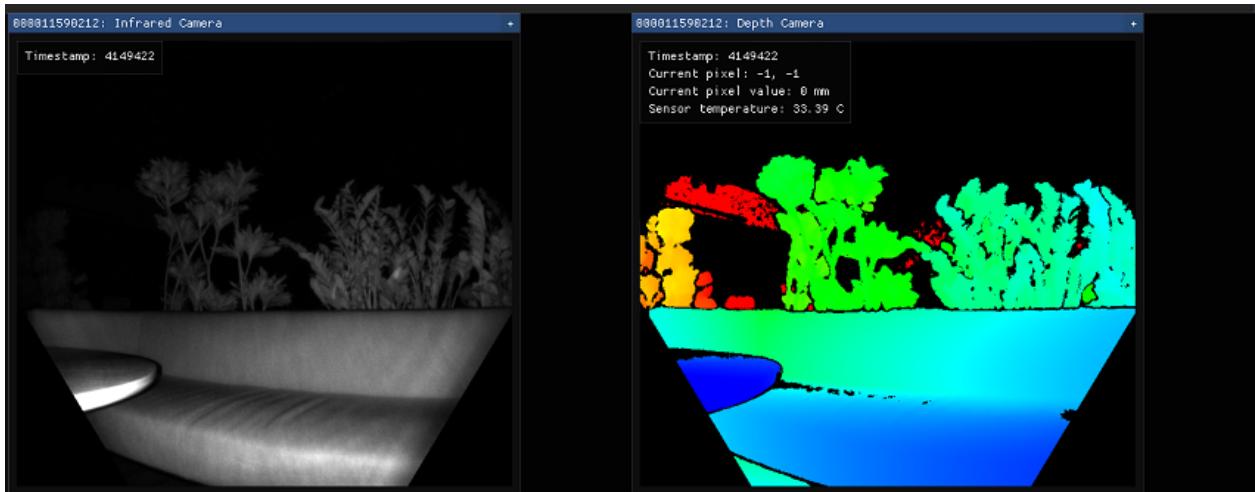
Par exemple, dans ce cas, le fournisseur d'accès Internet de la caméra de profondeur exécute FW 1.5.63.

Caméra à profondeur de champ

La visionneuse de la caméra de profondeur affiche deux fenêtres :

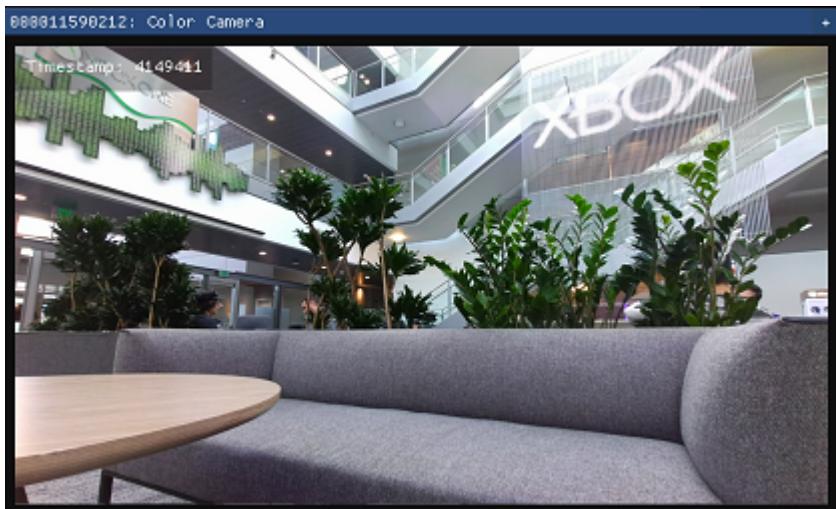
- L'une est appelée *Active Brightness* (Luminosité active), qui présente une image en nuances de gris montrant la luminosité de l'IR.
- L'autre est appelée *Depth* (Profondeur), qui présente une représentation en couleurs des données de profondeur.

Pointez le curseur sur le pixel de la fenêtre de profondeur pour afficher la valeur du capteur de profondeur, comme illustré ci-dessous.

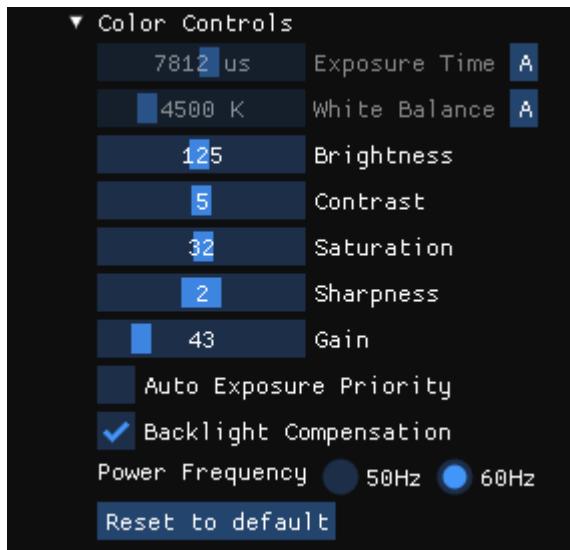


Caméra RVB

L'image ci-dessous montre l'affichage de la caméra couleur.



Vous pouvez contrôler les paramètres de caméra RVB à partir de la fenêtre de configuration pendant la diffusion en continu.

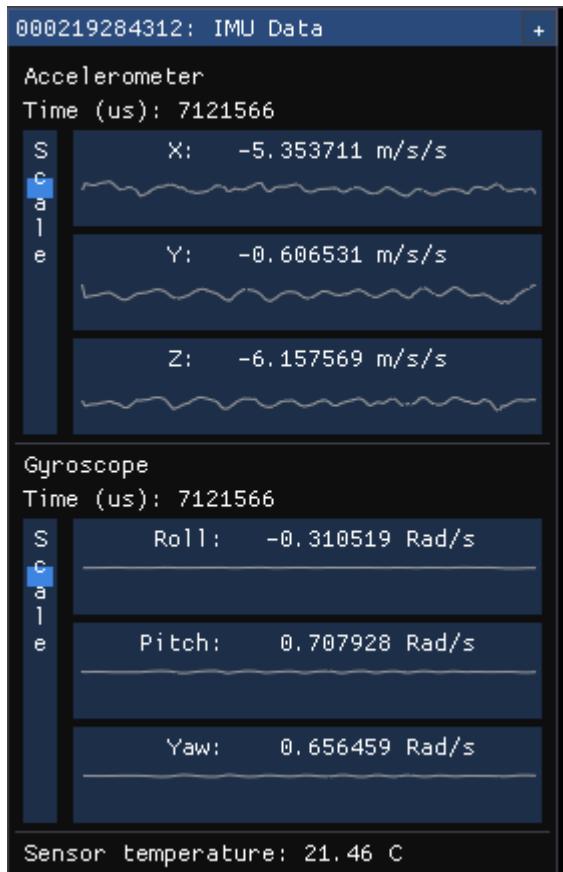


Unité de mesure inertielle (IMU)

La fenêtre IMU affiche deux composants : un accéléromètre et un gyroscope.

La moitié supérieure correspond à l'accéléromètre et montre l'accélération linéaire en mètres/seconde². Elle comprend l'accélération à partir de la gravité. Ainsi, mis à plat sur une table, l'axe Z affichera probablement une valeur avoisinant -9,8 m/s².

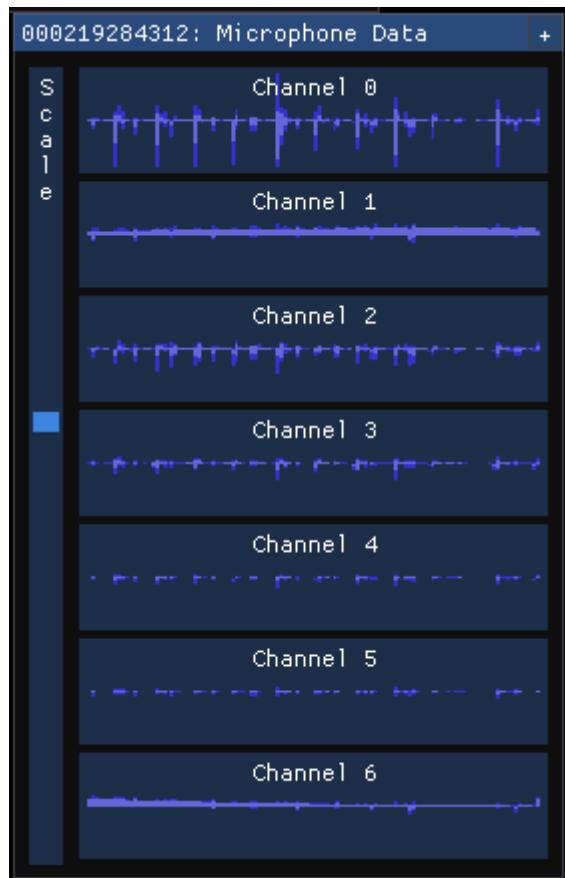
La moitié inférieure correspond au gyroscope et montre le mouvement de rotation en radians/seconde.



Entrée microphone

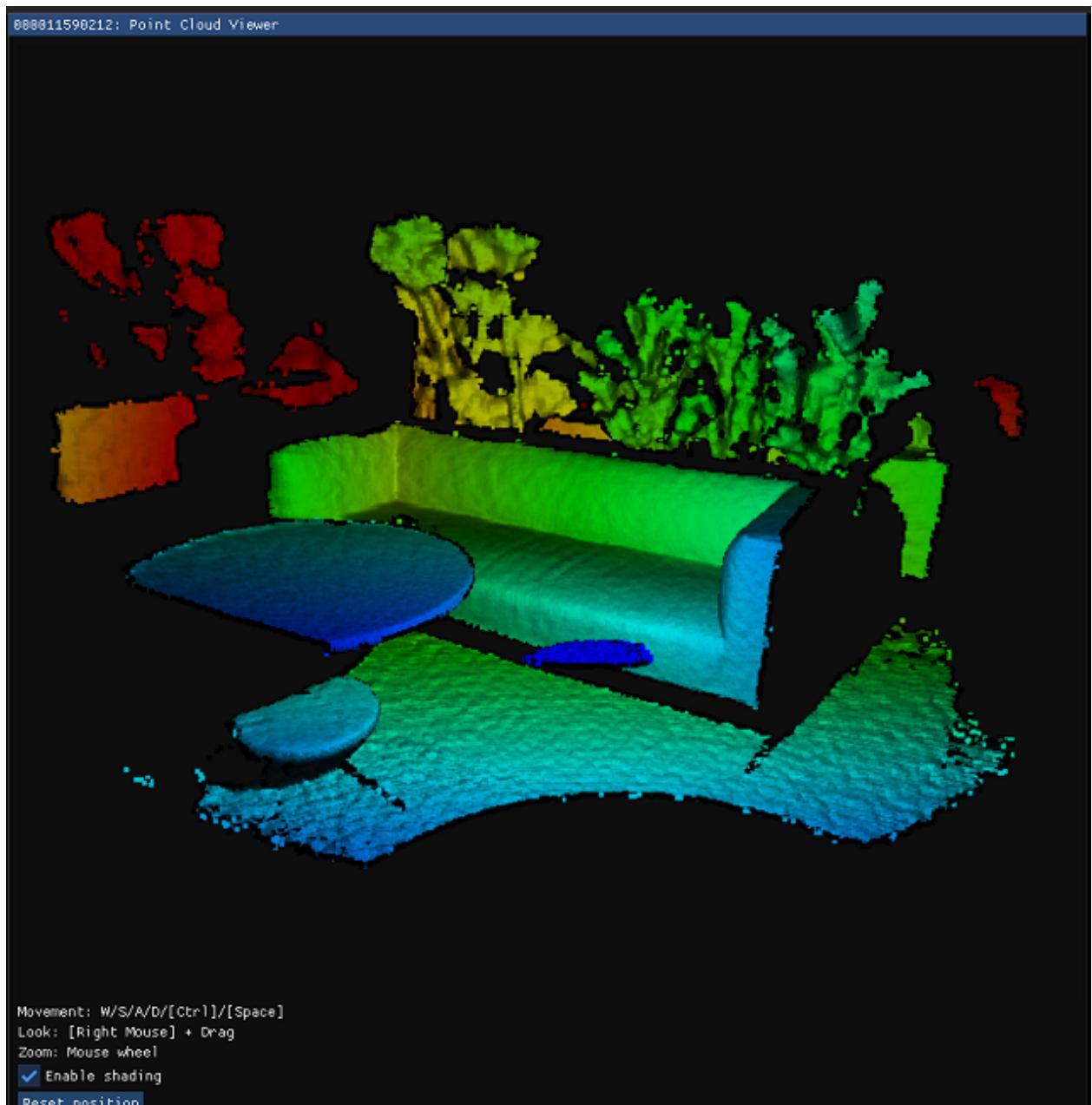
L'affichage du microphone représente le son audible sur chaque microphone. S'il n'y a pas de son, le graphique affiché est vide. Autrement, vous voyez une forme d'onde de couleur bleue foncée à laquelle est superposée une forme d'onde de couleur bleue claire.

L'onde foncée représente les valeurs minimales et maximales observées par le microphone pendant cette tranche horaire. L'onde claire représente la moyenne quadratique des valeurs observées par le microphone pendant cette tranche horaire.



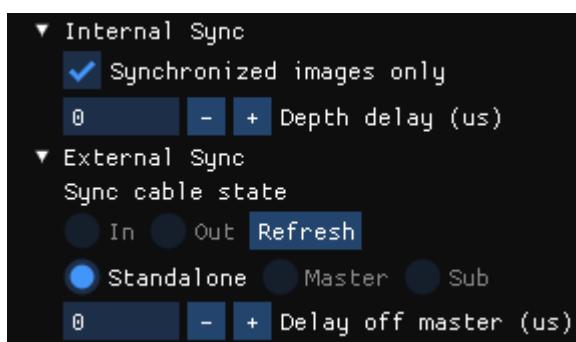
Visualisation de nuage de points

La profondeur visualisée en 3D vous permet de vous déplacer dans l'image à l'aide des touches.



Contrôle de synchronisation

Vous pouvez utiliser la visionneuse pour configurer l'appareil en mode autonome (par défaut), maître ou subordonné lors de la configuration de la synchronisation de plusieurs appareils. Lors de la modification de la configuration ou de l'insertion ou du retrait du câble de synchronisation, sélectionnez **Actualiser** pour mettre à jour.



Étapes suivantes

[Guide de configuration de la synchronisation externe](#)

Enregistreur Azure Kinect DK

Article • 01/06/2023

Cet article explique comment utiliser l'utilitaire de ligne de commande `k4arecorder` pour enregistrer des flux de données à partir du Kit de développement logiciel (SDK) du capteur dans un fichier.

ⓘ Notes

L'enregistreur Azure Kinect n'enregistre pas de signal audio.

Options de l'enregistreur

Le `k4arecorder` possède différents arguments de ligne de commande pour spécifier le fichier de sortie et les modes d'enregistrement.

Les enregistrements sont stockés au [format .mkv de Matroska](#). L'enregistrement utilise plusieurs pistes vidéo pour la couleur et la profondeur, ainsi que des informations supplémentaires telles que l'étalonnage et les métadonnées de la caméra.

Console

```
k4arecorder [options] output.mkv

Options:
  -h, --help                  Prints this help
  --list                      List the currently connected K4A devices
  --device                    Specify the device index to use (default: 0)
  -l, --record-length         Limit the recording to N seconds (default:
infinite)
  -c, --color-mode            Set the color sensor mode (default: 1080p),
Available options:
                                         3072p, 2160p, 1536p, 1440p, 1080p, 720p,
720p_NV12, 720p_YUY2, OFF
  -d, --depth-mode            Set the depth sensor mode (default:
NFOV_UNBINNED), Available options:
                                         NFOV_2X2BINNED, NFOV_UNBINNED, WFOV_2X2BINNED,
WFOV_UNBINNED, PASSIVE_IR, OFF
  --depth-delay               Set the time offset between color and depth frames
in microseconds (default: 0)
                                         A negative value means depth frames will arrive
before color frames.
                                         The delay must be less than 1 frame period.
  -r, --rate                  Set the camera frame rate in Frames per Second
                                         Default is the maximum rate supported by the
```

```
camera modes.
```

```
Available options: 30, 15, 5
```

```
--imu Set the IMU recording mode (ON, OFF, default: ON)  
--external-sync Set the external sync mode (Master, Subordinate,  
Standalone default: Standalone)
```

```
--sync-delay Set the external sync delay off the master camera  
in microseconds (default: 0)  
This setting is only valid if the camera is in  
Subordinate mode.
```

```
-e, --exposure-control Set manual exposure value (-11 to 1) for the RGB  
camera (default: auto exposure)
```

Fichiers d'enregistrement

Exemple 1. Enregistrer la profondeur en mode NFOV sans compartimentation (640x576), RVB 1080p à 30 i/s avec IMU. Appuyez sur les touches **CTRL-C** pour arrêter l'enregistrement.

```
k4arecorder.exe output.mkv
```

Exemple 2. Enregistrer WFOV sans compartimentation (1 MP), RVB 3072p à 15 i/s sans IMU, pendant 10 secondes.

```
k4arecorder.exe -d WFOV_UNBINNED -c 3072p -r 15 -l 10 --imu OFF output.mkv
```

Exemple 3. Enregistrer WFOV 2x2 avec compartimentation à 30 i/s pendant 5 secondes, puis sauvegarder dans le fichier output.mkv.

```
k4arecorder.exe -d WFOV_2X2BINNED -c OFF --imu OFF -l 5 output.mkv
```

💡 Conseil

Vous pouvez utiliser la **visionneuse Azure Kinect** pour configurer les contrôles de caméra RVB avant l'enregistrement (par exemple, pour régler la balance des blancs manuellement).

Vérifier l'enregistrement

Vous pouvez ouvrir le fichier output.mkv avec la [visionneuse Azure Kinect](#).

Pour extraire des pistes ou afficher les informations du fichier, des outils tels que `mkvinfo` sont disponibles dans le kit de ressources [MKVToolNix](#).

Étapes suivantes

[Utiliser l'enregistreur avec des unités synchronisées externes](#)

Outil de microprogramme Azure Kinect DK

Article • 01/06/2023

L'outil de microprogramme Azure Kinect permet d'interroger et de mettre à jour le microprogramme de l'appareil Azure Kinect DK.

Afficher les unités connectées

Vous pouvez obtenir la liste des appareils connectés à l'aide de l'option -l.

```
AzureKinectFirmwareTool.exe -l
```

```
Console

== Azure Kinect DK Firmware Tool ==
Found 2 connected devices:
0: Device "000036590812"
1: Device "000274185112"
```

Vérifier la version du microprogramme de l'appareil

Vous pouvez vérifier les versions actuelles du microprogramme du premier appareil connecté à l'aide de l'option -q, par exemple, `AzureKinectFirmwareTool.exe -q`.

```
Console

== Azure Kinect DK Firmware Tool ==
Device Serial Number: 000036590812
Current Firmware Versions:
    RGB camera firmware:      1.5.92
    Depth camera firmware:    1.5.66
    Depth config file:       6109.7
    Audio firmware:          1.5.14
    Build Config:            Production
    Certificate Type:        Microsoft
```

Si plusieurs appareils sont connectés, vous pouvez spécifier l'appareil que vous souhaitez interroger en ajoutant le numéro de série complet à la commande, par exemple :

```
AzureKinectFirmwareTool.exe -q 000036590812
```

Mettre à jour le microprogramme de l'appareil

L'utilisation la plus courante de cet outil est la mise à jour du microprogramme de l'appareil. Effectuez la mise à jour en appelant l'outil à l'aide de l'option `-u`. Une mise à jour du microprogramme peut prendre quelques minutes, en fonction des fichiers de microprogramme à mettre à jour.

Pour obtenir des instructions pas à pas sur la mise à jour du microprogramme, consultez [Mise à jour du microprogramme Azure Kinect](#).

```
AzureKinectFirmwareTool.exe -u firmware\AzureKinectDK_Fw_1.5.926614.bin
```

Si plusieurs appareils sont connectés, vous pouvez spécifier celui que vous souhaitez interroger en ajoutant son numéro de série complet à la commande.

```
AzureKinectFirmwareTool.exe -u firmware\AzureKinectDK_Fw_1.5.926614.bin  
000036590812
```

Réinitialisation de l'appareil

Vous pouvez réinitialiser un appareil Azure Kinect DK connecté à l'aide de l'option `-r` si vous devez faire passer l'appareil dans un état connu.

Si plusieurs appareils sont connectés, vous pouvez spécifier celui que vous souhaitez interroger en ajoutant son numéro de série complet à la commande.

```
AzureKinectFirmwareTool.exe -r 000036590812
```

Inspecter le microprogramme

L'inspection du microprogramme vous permet d'obtenir les informations de version à partir d'un fichier bin du microprogramme avant de mettre à jour un appareil réel.

```
AzureKinectFirmwareTool.exe -i firmware\AzureKinectDK_Fw_1.5.926614.bin
```

```
Console  
  
== Azure Kinect DK Firmware Tool ==  
Loading firmware package  
. . \ tools \ updater \ firmware \ AzureKinectDK_Fw_1.5.926614. bin.  
File size: 1228844 bytes
```

```
This package contains:  
RGB camera firmware: 1.5.92  
Depth camera firmware: 1.5.66  
Depth config files: 6109.7 5006.27  
Audio firmware: 1.5.14  
Build Config: Production  
Certificate Type: Microsoft  
Signature Type: Microsoft
```

Options de l'outil de mise à jour du microprogramme

Console

```
== Azure Kinect DK Firmware Tool ==  
* Usage Info *  
AzureKinectFirmwareTool.exe <Command> <Arguments>  
  
Commands:  
List Devices: -List, -l  
Query Device: -Query, -q  
    Arguments: [Serial Number]  
Update Device: -Update, -u  
    Arguments: <Firmware Package Path and FileName> [Serial Number]  
Reset Device: -Reset, -r  
    Arguments: [Serial Number]  
Inspect Firmware: -Inspect, -i  
    Arguments: <Firmware Package Path and FileName>  
  
If no Serial Number is provided, the tool will just connect to the first device.  
  
Examples:  
AzureKinectFirmwareTool.exe -List  
AzureKinectFirmwareTool.exe -Update c:\data\firmware.bin 0123456
```

Étapes suivantes

Instructions pas à pas pour mettre à jour le microprogramme d'un appareil

Télécharger le Kit de développement logiciel (SDK) du capteur Azure Kinect

Article • 01/06/2023

Cette page reprend les liens de téléchargement pour chaque version du Kit de développement logiciel (SDK) du capteur Azure Kinect. Le programme d'installation fournit tous les fichiers nécessaires à développer pour Azure Kinect.

Contenus du Kit de développement logiciel (SDK) du capteur Azure Kinect

- En-têtes et bibliothèques pour créer une application à l'aide de l'Azure Kinect DK.
- DLL redistribuables requises par les applications utilisant l'Azure Kinect DK.
- La [Visionneuse Azure Kinect](#).
- L'[Enregistreur Azure Kinect](#).
- L'[Outil du microprogramme Azure Kinect](#).

Instructions d'installation Windows

Vous trouverez des détails sur l'installation des versions les plus récentes et précédentes du Kit de développement logiciel (SDK) et du microprogramme du capteur Azure Kinect [ici ↗](#).

Vous trouverez le fichier source [ici ↗](#).

ⓘ Notes

Lorsque vous installez le Kit de développement logiciel (SDK), rappelez-vous le chemin d'installation. Par exemple, « C:\Program Files\Azure Kinect SDK 1.2 ». Vous trouverez les outils référencés dans les articles à ce chemin.

Instructions d'installation Linux

Actuellement, la seule distribution prise en charge est Ubuntu 18.04. Pour demander la prise en charge d'autres distributions, consultez [cette page ↗](#).

Tout d'abord, vous devez configurer le [dépôt du package Microsoft](#), en suivant les instructions reprises [ici](#).

Vous pouvez maintenant installer les packages nécessaires. Le `k4a-tools` package inclut la [Visionneuse Azure Kinect](#), l'[enregistreur Azure Kinect](#) et l'[outil du microprogramme Azure Kinect](#). Pour installer le package, exécutez :

```
sudo apt install k4a-tools
```

Cette commande installe les packages de dépendances nécessaires au bon fonctionnement des outils, notamment la dernière version de `libk4a<major>.<minor>`.

Vous devez ajouter des règles udev pour pouvoir accéder à Azure Kinect DK si vous n'êtes pas l'utilisateur racine. Pour obtenir des instructions, consultez [Linux Device Setup](#). Vous pouvez également lancer des applications qui utilisent l'appareil en tant que racine.

Le package `libk4a<major>.<minor>-dev` contient les en-têtes et les fichiers CMake permettant de créer des applications/exécutables par rapport à `libk4a`.

Le package `libk4a<major>.<minor>` contient les objets partagés nécessaires à l'exécution des applications/exécutables qui dépendent de `libk4a`.

Les didacticiels de base nécessitent le package `libk4a<major>.<minor>-dev`. Pour installer le package, exécutez :

```
sudo apt install libk4a<major>.<minor>-dev
```

Si la commande est réussie, le Kit de développement logiciel (SDK) est prêt à être utilisé.

Veillez à installer la version correspondante de `libk4a<major>.<minor>` avec `libk4a<major>.<minor>-dev`. Par exemple, si vous installez le package `libk4a4.1-dev`, installez le package `libk4a4.1` correspondant qui contient la version correspondante des fichiers objets partagés. Pour obtenir la dernière version de `libk4a`, consultez les liens dans la section suivante.

Journal des modifications et versions antérieures

Vous trouverez le journal des modifications pour le Kit de développement logiciel (SDK) [ici](#).

Si vous avez besoin d'une version antérieure du Kit de développement logiciel (SDK) du capteur Azure Kinect, recherchez-la [ici](#).

Étapes suivantes

[Configurer Azure Kinect DK](#)

Télécharger le Kit de développement logiciel (SDK) de suivi de corps d'Azure Kinect

Article • 01/06/2023

Ce document fournit des liens pour installer chaque version du Kit de développement logiciel (SDK) de suivi de corps d'Azure Kinect.

Contenu du Kit de développement logiciel (SDK) de suivi de corps d'Azure Kinect

- En-têtes et bibliothèques pour créer une application à l'aide de la solution Azure Kinect DK.
- DLL redistribuables requises par les applications de suivi de corps utilisant la solution Azure Kinect DK.
- Exemples d'applications de suivi de corps.

Liens de téléchargement Windows

| Version | Téléchargement |
|---------|---|
| 1.1.2 | msi ↗ nuget ↗ |
| 1.1.1 | msi ↗ nuget ↗ |
| 1.1.0 | msi ↗ |
| 1.0.1 | msi ↗ nuget ↗ |
| 1.0.0 | msi ↗ nuget ↗ |

Instructions d'installation Linux

Actuellement, la seule distribution prise en charge est Ubuntu 18.04 et 20.04. Pour demander la prise en charge d'autres distributions, consultez [cette page ↗](#).

Tout d'abord, vous devez configurer le [dépôt du package Microsoft ↗](#), en suivant les instructions reprises [ici](#).

Le package `libk4abt<major>.<minor>-dev` contient les en-têtes et les fichiers CMake à créer par rapport à `libk4abt`. Le package `libk4abt<major>.<minor>` contient les objets partagés nécessaires pour exécuter les exécutables qui dépendent de `libk4abt`, ainsi que l'exemple de visionneuse.

Les didacticiels de base nécessitent le package `libk4abt<major>.<minor>-dev`. Pour l'installer, exécutez

```
sudo apt install libk4abt<major>.<minor>-dev
```

Si la commande est réussie, le Kit de développement logiciel (SDK) est prêt à être utilisé.

ⓘ Notes

Lorsque vous installez le Kit de développement logiciel (SDK), rappelez-vous le chemin d'installation. Par exemple, « C:\Program Files\Azure Kinect Body Tracking SDK 1.0.0 ». Les exemples référencés dans les articles sont accessibles via ce chemin. Les exemples de suivi de corps se trouvent dans le dossier **Body-Tracking-Samples** dans le référentiel Azure-Kinect-Samples. Les exemples référencés dans les articles se trouvent ici.

Journal des modifications

v1.1.2

- [Fonctionnalité] Ajout de la prise en charge du wrapper C# pour Linux [Lien ↗](#)
- [Correctif de bogue] `k4abt_simple_3d_viewer.exe` fonctionne avec les derniers pilotes NVIDIA [Lien ↗](#)

v1.1.1

- [Fonctionnalité] Ajout de la prise en charge de cmake à tous les exemples de suivi de corps
- [Fonctionnalité] Le package NuGet retourne. Développement d'un nouveau package NuGet incluant des dll et en-têtes de suivi de corps développés par Microsoft, ainsi que des dépendances de runtime ONNX. Le package n'inclut plus les dépendances NVIDIA CUDA et TRT. Celles-ci restent incluses dans le package MSI.
- [Fonctionnalité] Mise à niveau vers le runtime ONNX v1.10. La version recommandée du pilote NVIDIA est 472.12 (Prêt pour le jeu) ou 472.84 (Studio). Il

existe des problèmes OpenGL avec les pilotes ultérieurs.

- [Correctif de bogue] CMake manquant dans l'exemple offline_processor [Lien ↗](#)
- [Correctif de bogue] Le mode UC ne requiert plus de [Lien ↗](#) de dépendances NVIDIA CUDA
- [Correctif de bogue] Les exemples vérifiés compilent avec Visual Studio 2022 et les exemples mis à jour pour utiliser cette version [Lien ↗](#)
- [Correctif de bogue] Ajout d'un qualificateur const aux API [Lien ↗](#)
- [Correctif de bogue] Ajout de la vérification du descripteur nullptr dans shutdown() [Lien ↗](#)
- [Correctif de bogue] Vérifications des dépendances améliorées [Lien ↗](#)
- [Correctif de bogue] Mise à jour du fichier REDIST.TXT [Lien ↗](#)
- [Correctif de bogue] Amélioration des performances de DirectML [Lien ↗](#)
- [Correctif de bogue] Correction de la déclaration d'exception dans frame::get_body() [Lien ↗](#)
- [Correctif de bogue] Correction de fuite de mémoire [Lien ↗](#)
- [Correctif de bogue] Mise à jour de la liste des dépendances [Lien ↗](#)

v1.1.0

- [Fonctionnalité] Ajout de la prise en charge de l'exécution DirectML (Windows uniquement) et TensorRT du modèle d'estimation de pose. Consultez le Forum aux questions sur les nouveaux environnements d'exécution.
- [Fonctionnalité] Ajout de `model_path` à la structure `k4abt_tracker_configuration_t`. Permet aux utilisateurs de spécifier le chemin d'accès pour le modèle d'estimation de pose. La valeur par défaut est le modèle d'estimation de pose standard `dnn_model_2_0_op11.onnx` situé dans le répertoire actif.
- [Fonctionnalité] Incluez le modèle d'estimation de pose `dnn_model_2_0_lite_op11.onnx` lite. Ce modèle stocke est environ 2 fois plus rapide au détriment d'environ 5 % de précision.
- [Fonctionnalité] Les exemples vérifiés compilent avec Visual Studio 2019 et mettent à jour les exemples pour utiliser cette version.
- [Changement cassant] Mise à jour vers les environnements d'exécution ONNX Runtime 1.6 avec prise en charge du processeur, CUDA 11.1, DirectML (Windows uniquement) et TensorRT 7.2.1. Nécessite la mise à jour du pilote Nvidia vers R455 ou une version ultérieure.
- [Changement cassant] Aucune installation de NuGet.
- [Correctif de bogue] Ajout la prise en charge des GPU de la série Nvidia RTX 30xx [Lien ↗](#)
- [Correctif de bogue] Ajout de la prise en charge des GPU intégrés AMD et Intel (Windows uniquement) [Lien ↗](#)

- [Correctif de bogue] Mise à jour vers CUDA 11.1 [Lien ↗](#)
- [Correctif de bogue] Mise à jour vers Sensor SDK 1.4.1 [Lien ↗](#)

v1.0.1

- [Correctif de bogue] Résolution du problème de blocage du Kit de développement logiciel (SDK) lors du chargement d'onnxruntime. dll à partir du chemin d'accès sur Windows Build 19025 ou version ultérieure : [Lien ↗](#)

v1.0.0

- [Fonctionnalité] Ajout d'un wrapper C# au programme d'installation msi.
- [Correctif de bogue] Résolution du problème de non-détection correcte de la rotation la tête : [Lien ↗](#)
- [Correctif de bogue] Résolution du problème d'utilisation de l'UC jusqu'à 100 % sur une machine Linux : [Lien ↗](#)
- [Exemples] Ajout de deux exemples au référentiel. L'exemple 1 montre comment transformer les résultats de suivi de corps de l'espace de profondeur en espace de couleurs [Lien ↗](#) ; l'exemple 2 montre comment détecter un plan de sol [Lien ↗](#)

Étapes suivantes

- [Vue d'ensemble du kit Azure Kinect DK](#)
- [Configurer Azure Kinect DK](#)
- [Configurer le suivi des corps Azure Kinect](#)

Configuration requise pour le kit SDK Azure Kinect Sensor

Article • 01/06/2023

Ce document fournit des informations sur la configuration requise pour installer le kit SDK du capteur et déployer correctement le kit Azure Kinect DK.

Systèmes d'exploitation et architectures pris en charge

- Windows 10 avril 2018 (version 1803, build d'OS 17134) x64 ou version ultérieure
- Linux Ubuntu 18.04 x64 avec pilote de GPU utilisant OpenGL4.4 ou version ultérieure

Le kit SDK Sensor est disponible pour l'API Windows (Win32) des applications Windows C/C++ natives. Le kit SDK n'est pas disponible pour les applications UWP. Azure Kinect DK n'est pas pris en charge pour Windows 10 en mode S.

Exigences relatives à l'environnement de développement

Pour contribuer au développement du kit SDK du capteur, accédez à [GitHub](#).

Configuration matérielle minimale du PC hôte

La configuration matérielle requise sur le PC hôte dépend de l'application, de l'algorithme, de la fréquence d'images du capteur et de la résolution disponibles sur le PC hôte. Configuration minimale recommandée pour le kit SDK du capteur sur Windows :

- Processeur Intel® Core™ i3 de septième génération (double cœur 2,4 GHz avec GPU HD620 ou supérieur)
- 4 Go de mémoire
- Port USB3 dédié
- Prise en charge du pilote graphique pour OpenGL 4.4 ou DirectX 11.0

Les processeurs d'entrée de gamme ou plus anciens peuvent également fonctionner selon votre cas d'usage.

Les performances diffèrent également entre les systèmes d'exploitation Windows/Linux et les pilotes graphiques utilisés.

Configuration matérielle du PC hôte de suivi des corps

La configuration d'un PC hôte de suivi des corps est plus stricte que la configuration d'un PC hôte général. Configuration minimale recommandée pour le kit SDK de suivi des corps sur Windows :

- Processeur Intel® CoreTM i5 de septième génération (quadricœur 2,4 GHz ou plus rapide)
- 4 Go de mémoire
- NVIDIA GEFORCE GTX 1050 ou équivalent
- Port USB3 dédié

La configuration minimale recommandée suppose un mode de profondeur K4A_DEPTH_MODE_NFOV_UNBINNED à 30 images par seconde pour le suivi de 5 personnes. Les processeurs ainsi que les GPU NVIDIA d'entrée de gamme ou plus anciens peuvent également fonctionner selon votre cas d'usage.

USB3

Il existe des problèmes de compatibilité connus avec les contrôleurs hôtes USB. Pour plus d'informations, consultez la [page de résolution des problèmes](#)

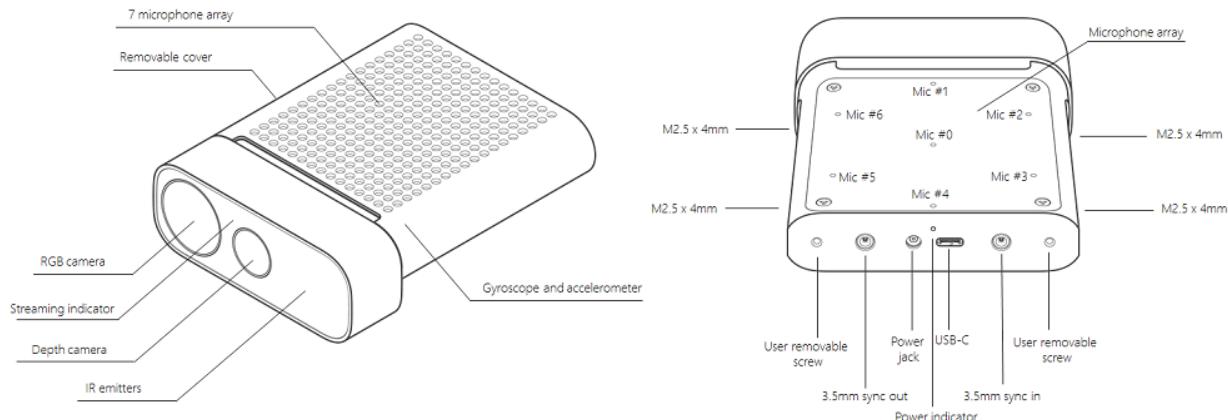
Étapes suivantes

- [Vue d'ensemble du kit Azure Kinect DK](#)
- [Configurer Azure Kinect DK](#)
- [Configurer le suivi des corps Azure Kinect](#)

Spécifications matérielles pour Azure Kinect DK

Article • 01/06/2023

Cet article fournit des informations sur la façon dont le matériel Azure Kinect intègre la toute dernière technologie de capteur de Microsoft dans un accessoire connecté par USB.



Termes

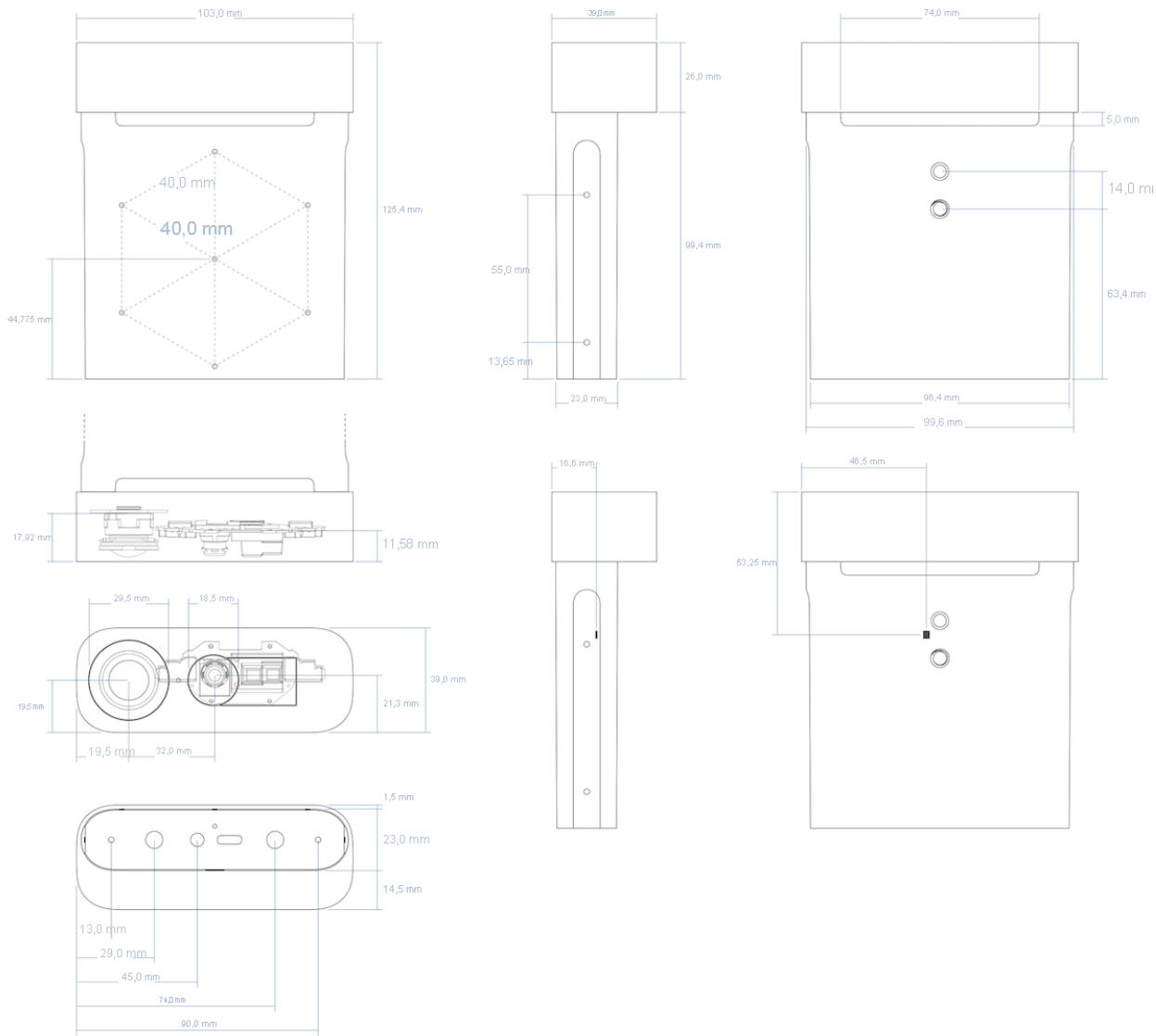
Les abréviations suivantes seront utilisées tout au long de cet article.

- NFOV (mode de profondeur à champ de vision étroit)
- WFOV (mode de profondeur à champ de vision large)
- FOV (champ de vision)
- FPS (images par seconde)
- IMU (unité de mesure inertielle)
- FoI (champ d'intérêt)

Dimensions et poids du produit

Le poids et les dimensions de l'appareil Azure Kinect sont les suivants.

- **Dimensions** : 103 x 39 x 126 mm
- **Poids** : 440 g



Un fichier STEP pour l'appareil Azure Kinect est disponible [ici](#).

Environnement d'exploitation

Azure Kinect DK est destiné aux développeurs et aux entreprises commerciales disposant des conditions ambiantes suivantes :

- **Température** : 10-25°C
- **Humidité** : 8 à 90 % d'humidité relative (sans condensation)

① Notes

L'utilisation de cet appareil dans des conditions ambiantes autres que celles-ci peut provoquer la panne de l'appareil ou un fonctionnement anormal. Ces conditions ambiantes sont applicables à l'environnement immédiat de l'appareil, quelles que soient les conditions opérationnelles. Lorsque l'appareil est utilisé avec un boîtier externe, il est recommandé d'utiliser une solution de contrôle actif de la

température et/ou d'autres solutions de refroidissement afin de garantir le maintien de la température dans les plages indiquées. L'appareil comprend un canal de refroidissement situé entre la section avant et la section arrière. Lorsque vous utilisez l'appareil, vérifiez que le canal de refroidissement n'est pas obstrué.

Reportez-vous aux [informations de sécurité](#) supplémentaires sur le produit.

Modes de fonctionnement pris en charge pour la caméra à profondeur de champ

Azure Kinect DK comprend une caméra à profondeur de champ ToF 1 Mpx conçue par Microsoft qui utilise le [capteur d'images présenté lors de la conférence ISSCC 2018](#). La caméra à profondeur de champ prend en charge les modes indiqués ci-dessous :

| Mode | Résolution | FoV | i/s | Plage de fonctionnement* | Temps d'exposition |
|--------------------------------------|-------------|-----------|--------------------|--------------------------|--------------------|
| NFOV sans compartimentation | 640 x 576 | 75°x°65 | 0, 5, 15, 30 | 0,5 - 3,86 m | 12,8 ms |
| NFOV 2x2 avec compartimentation (SW) | 320 x 288 | 75°x°65 | 0, 5, 15, 30 | 0,5 - 5,46 m | 12,8 ms |
| WFOV 2x2 avec compartimentation | 512 x 512 | 120°x°120 | 0, 5, 15, 30 | 0,25 - 2,88 m | 12,8 ms |
| WFOV sans compartimentation | 1024 x 1024 | 120°x°120 | 0, 5, 15 | 0,25 - 2,21 m | 20,3 ms |
| Infrarouge passif | 1024 x 1024 | N/A | 0, 5, 15, 30 | N/A | 1,6 ms |

Réflectivité de 15 à 95 % à 850 nm, 2,2 μ W/cm²/nm, erreur aléatoire écart type. \leq 17 mm, erreur systématique typique < 11 mm + 0,1 % de la distance sans interférence multichemin. Une profondeur peut être fournie en dehors de la plage de fonctionnement indiquée ci-dessus. Elle dépend de la réflectivité d'un objet.

Modes de fonctionnement pris en charge pour la caméra couleur

Azure Kinect DK comprend un capteur CMOS OV12A10 12 Mpx avec obturateur roulant. Les modes de fonctionnement natifs sont listé ci-dessous :

| Résolution de la caméra RVB (H x V) | Proportions | Options de format | Fréquence d'images (FPS) | FOV nominal (H x V) (après traitement) |
|-------------------------------------|-------------|-------------------|--------------------------|--|
| 3 840 x 2 160 | 16:9 | MJPEG | 0, 5, 15, 30 | 90°x°59 |
| 2 560 x 1 440 | 16:9 | MJPEG | 0, 5, 15, 30 | 90°x°59 |
| 1 920 x 1 080 | 16:9 | MJPEG | 0, 5, 15, 30 | 90°x°59 |
| 1 280 x 720 | 16:9 | MJPEG/YUY2/NV12 | 0, 5, 15, 30 | 90°x°59 |
| 4 096 x 3 072 | 4:3 | MJPEG | 0, 5, 15 | 90°x°74,3 |
| 2 048 x 1 536 | 4:3 | MJPEG | 0, 5, 15, 30 | 90°x°74,3 |

La caméra RVB est compatible avec les classes vidéo USB et peut être utilisée sans le SDK du capteur. Espace colorimétrique de la caméra RVB : plage complète BT.601 [0..255]. Le [sous-échantillonnage de la chrominance](#) ↗ MJPEG est 4:2:2.

ⓘ Notes

Le SDK du capteur peut fournir des images en couleurs au format de pixel BVRA. Ce mode n'est pas pris en charge par l'appareil et provoque une charge processeur supplémentaire lorsqu'il est utilisé. Le processeur hôte est utilisé pour convertir les images JPEG reçues de l'appareil.

Valeurs de temps d'exposition de la caméra RVB

Vous trouverez ci-dessous les valeurs d'exposition manuelle qui sont acceptées pour la caméra RVB :

| exp | 2^exp | 50 Hz | 60 Hz |
|-----|-------|-------|-------|
| -11 | 488 | 500 | 500 |
| -10 | 977 | 1250 | 1250 |
| -9 | 1953 | 2 500 | 2 500 |
| -8 | 3906 | 10000 | 8330 |

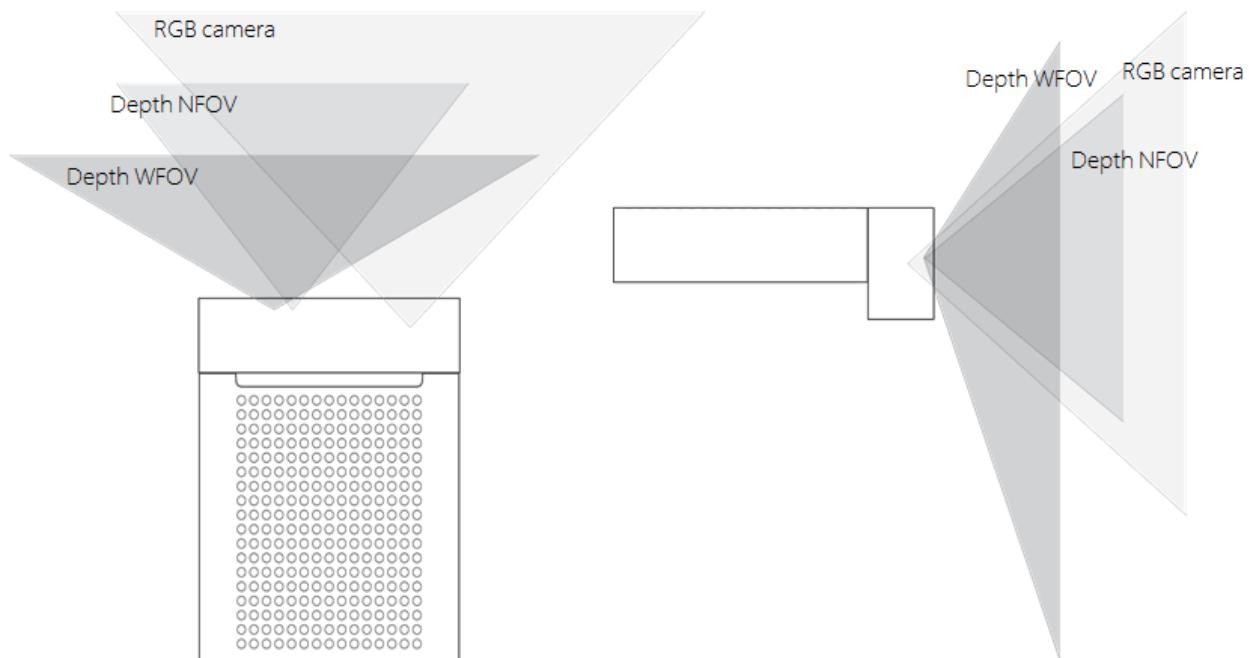
| exp | 2^{exp} | 50 Hz | 60 Hz |
|------------|------------------------|--------------|--------------|
| -7 | 7813 | 20000 | 16670 |
| -6 | 15625 | 30000 | 33330 |
| -5 | 31250 | 40000 | 41670 |
| -4 | 62500 | 50000 | 50000 |
| -3 | 125000 | 60000 | 66670 |
| -2 | 250 000 | 80000 | 83330 |
| -1 | 500000 | 100000 | 100000 |
| 0 | 1000000 | 120 000 | 116670 |
| 1 | 2000000 | 130000 | 133330 |

Minutage brut du capteur de profondeur

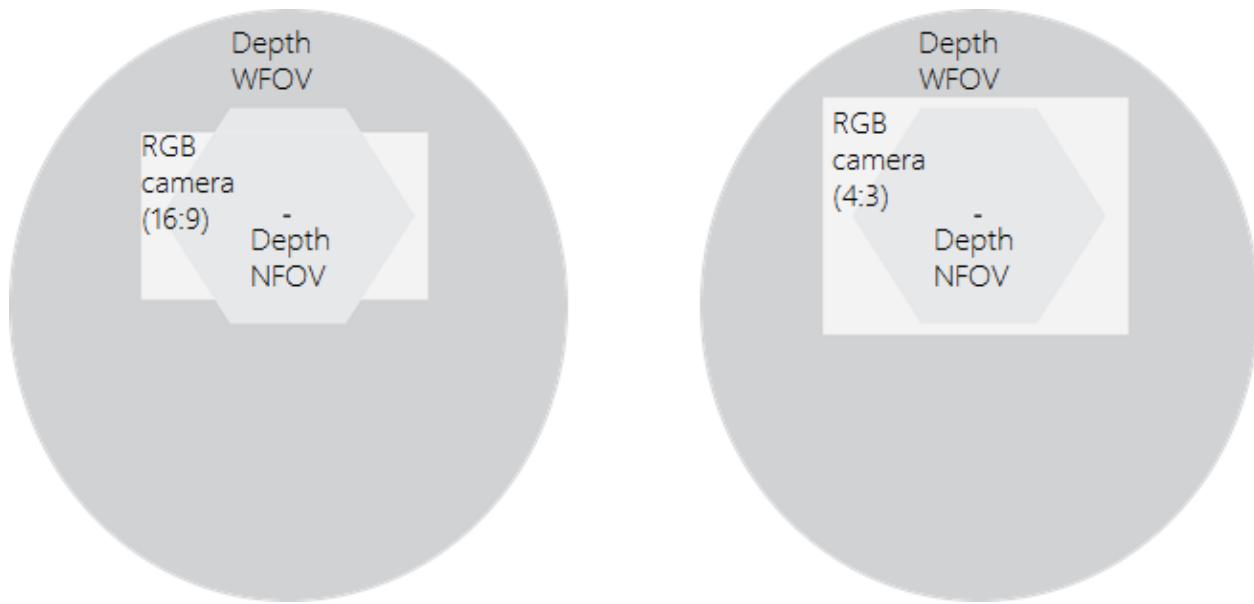
| Mode de profondeur | IR Impulsions | d'impulsion Largeur | Idle Points | Durée d'inactivité | Exposition : Temps |
|------------------------------------|--------------------------|--------------------------------|------------------------|-------------------------------|-----------------------------------|
| NFOV sans compartimentation | 9 | 125 us | 8 | 1 450 us | 12,8 ms |
| NFOV 2xx avec compartimentation | | | | | |
| WFOV 2x2 avec compartimentation | | | | | |
| WFOV sans compartimentation | 9 | 125 us | 8 | 2 390 us | 20,3 ms |

Champ de vision de la caméra

L'image suivante montre la profondeur et le champ de vision de la caméra RVB, ou les angles que les capteurs « voient ». Ce diagramme montre la caméra RVB en mode 4:3.



Cette image montre le champ de vision de la caméra, vu de devant à une distance de 2 000 mm.



ⓘ Notes

Lorsque la profondeur est en mode NFOV, la caméra RVB dispose d'un meilleur chevauchement des pixels avec une résolution 4:3 qu'avec une résolution 16:9.

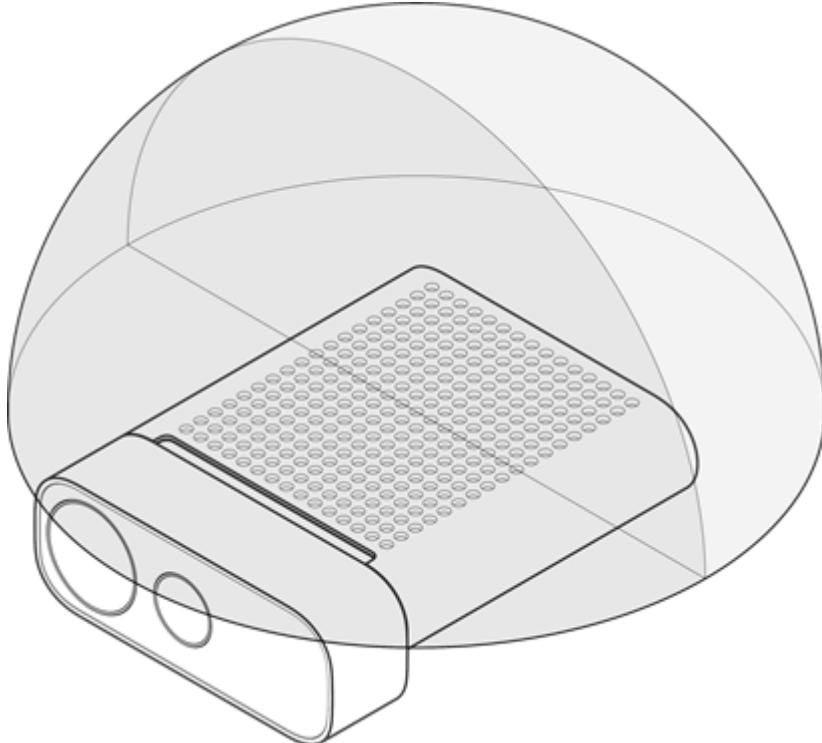
Capteur de mouvement (IMU)

L'unité de mesure inertielle (IMU) incorporée est un LSM6DSMUS qui comprend un accéléromètre et un gyroscope. L'accéléromètre et le gyroscope sont échantillonnés simultanément à 1,6 kHz. Les échantillons sont signalés à l'hôte à 208 Hz.

Réseau de microphones

Azure Kinect DK comprend un réseau circulaire de 7 microphones haute qualité qui correspond à un appareil de classe audio USB 2.0 standard. Vous pouvez accéder à l'ensemble des 7 canaux. Les spécifications de performances sont les suivantes :

- Sensibilité : -22 dBFS (94 dB SPL, 1 kHz)
- Rapport signal/bruit > 65 dB
- Point de surcharge acoustique : 116 dB



USB

Azure Kinect DK est un appareil composite USB3 qui expose les points de terminaison matériels suivants au système d'exploitation :

L'ID de fournisseur est 0x045E (Microsoft). La table Product ID est montrée ci-dessous :

| Interface USB | IP PNP | Notes |
|------------------------------|--------|---------------|
| Hub USB 3.1 Gen2 | 0x097A | Hub principal |
| Hub USB 2.0 | 0x097B | HS USB |
| Caméra à profondeur de champ | 0x097C | USB 3.0 |
| Caméra couleur | 0x097D | USB 3.0 |
| Microphones | 0x097E | HS USB |

Indicateurs

L'appareil possède un voyant pour la diffusion de données de la caméra, qui est situé à l'avant et qui peut être désactivé par programmation à l'aide du SDK du capteur.

Le voyant situé à l'arrière de l'appareil indique l'état de celui-ci :

| Si le voyant est | Signification |
|-------------------------------|---|
| Blanc et fixe | L'appareil est allumé et fonctionne correctement. |
| Blanc clignotant | L'appareil est allumé mais ne dispose pas d'une connexion de données USB 3.0. |
| Orange clignotant | L'appareil n'a pas suffisamment de puissance pour fonctionner. |
| Orange, puis blanc clignotant | Mise à jour du microprogramme ou récupération en cours |

Bloc d'alimentation

L'appareil peut être alimenté de deux manières :

1. À l'aide du dispositif d'alimentation fourni avec l'appareil. Le connecteur d'alimentation est un OD 4,5 mm avec un ID 3,0 mm et un diamètre de broche de 0,6 mm.
2. En utilisant un câble Type C vers Type C à la fois pour l'alimentation et pour les données.

L'appareil Azure Kinect DK n'est pas fourni avec un câble Type C vers Type C.

ⓘ Notes

- Le câble d'alimentation fourni est un câble USB Type A vers connecteur cylindrique simple. Utilisez le dispositif d'alimentation murale fourni avec ce câble. L'appareil est capable de prendre plus d'énergie que deux ports USB Type A standard ne peuvent fournir.
- Les câbles USB sont très importants. Nous vous recommandons donc d'utiliser des câbles de haute qualité et de vérifier qu'ils fonctionnent correctement avant de déployer l'unité à distance.

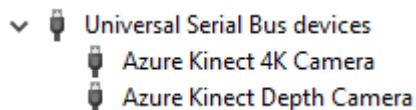
💡 Conseil

Pour sélectionner un bon câble Type C vers Type C :

- Le **câble certifié USB** doit prendre en charge à la fois l'alimentation et les données.
- Un câble passif doit mesurer moins de 1,5 mètre. S'il est plus long, utilisez plutôt un câble actif.
- Le câble doit prendre en charge au moins 1,5 A. Dans le cas contraire, vous devrez connecter une alimentation externe.

Vérifiez le câble :

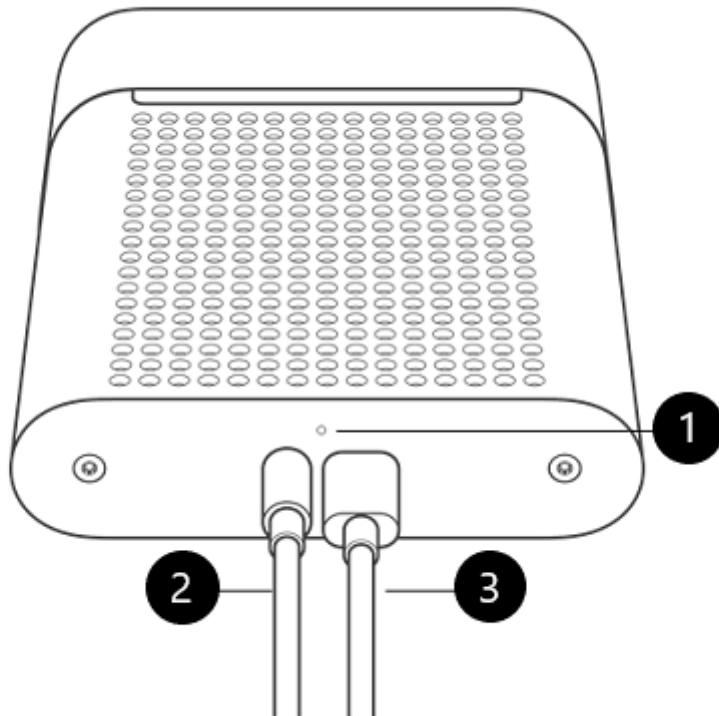
- Connectez l'appareil via le câble au PC hôte.
- Vérifiez que tous les appareils sont énumérés correctement dans le gestionnaire de périphériques Windows. La caméra à profondeur de champ et la caméra RVB doivent s'afficher comme dans l'exemple ci-dessous.



- Dans la visionneuse Azure Kinect, vérifiez que le câble peut diffuser des données de façon fiable pour tous les capteurs, avec les paramètres suivants :
 - Caméra à champ de profondeur : NFOV unbinned
 - Caméra RVB : 2160p
 - Microphones et IMU activés

Que signifie ce voyant ?

Le voyant d'alimentation est situé à l'arrière de l'appareil Azure Kinect DK. La couleur du voyant change en fonction de l'état de l'appareil.



Ce schéma montre les composants suivants :

1. Le voyant d'alimentation
2. Le câble d'alimentation (connecté à une source d'alimentation)
3. Le câble de données USB Type C (connecté au PC)

Vérifiez que les câbles sont connectés comme indiqué. Consultez ensuite le tableau suivant pour connaître les différents états du voyant d'alimentation.

| Si le voyant est : | Signification : | Vous devez : |
|---------------------------|---|---|
| Blanc et fixe | L'appareil est sous tension et fonctionne correctement. | Utiliser l'appareil. |
| Éteint | L'appareil n'est pas connecté au PC. | Vérifier que le câble du connecteur d'alimentation rond est connecté à l'appareil et à l'adaptateur secteur USB. Vérifier que le câble USB Type C est connecté à l'appareil et à votre PC. |

| Si le voyant est : | Signification : | Vous devez : |
|-------------------------------|---|--|
| Blanc clignotant | L'appareil est allumé mais ne dispose pas d'une connexion de données USB 3.0. | <p>Vérifier que le câble du connecteur d'alimentation rond est connecté à l'appareil et à l'adaptateur secteur USB.</p> <p>Vérifier que le câble USB Type C est connecté à l'appareil et à un port USB 3.0 du PC.</p> <p>Connecter l'appareil sur un autre port USB 3.0 du PC.</p> <p>Sur l'ordinateur, ouvrir le Gestionnaire de périphériques (Démarrer>Panneau de configuration>Gestionnaire de périphériques) et vérifier que l'ordinateur dispose d'un contrôleur hôte USB 3.0 pris en charge.</p> |
| Orange clignotant | L'appareil n'a pas suffisamment de puissance pour fonctionner. | <p>Vérifier que le câble du connecteur d'alimentation rond est connecté à l'appareil et à l'adaptateur secteur USB.</p> <p>Vérifier que le câble USB Type C est connecté à l'appareil et à votre PC.</p> |
| Orange, puis blanc clignotant | L'appareil est sous tension et reçoit une mise à jour du microprogramme ou restaure les paramètres d'usine. | Attendre que le voyant d'alimentation soit blanc et fixe. Pour plus d'informations, consultez Réinitialiser Azure Kinect DK . |

Consommation énergétique

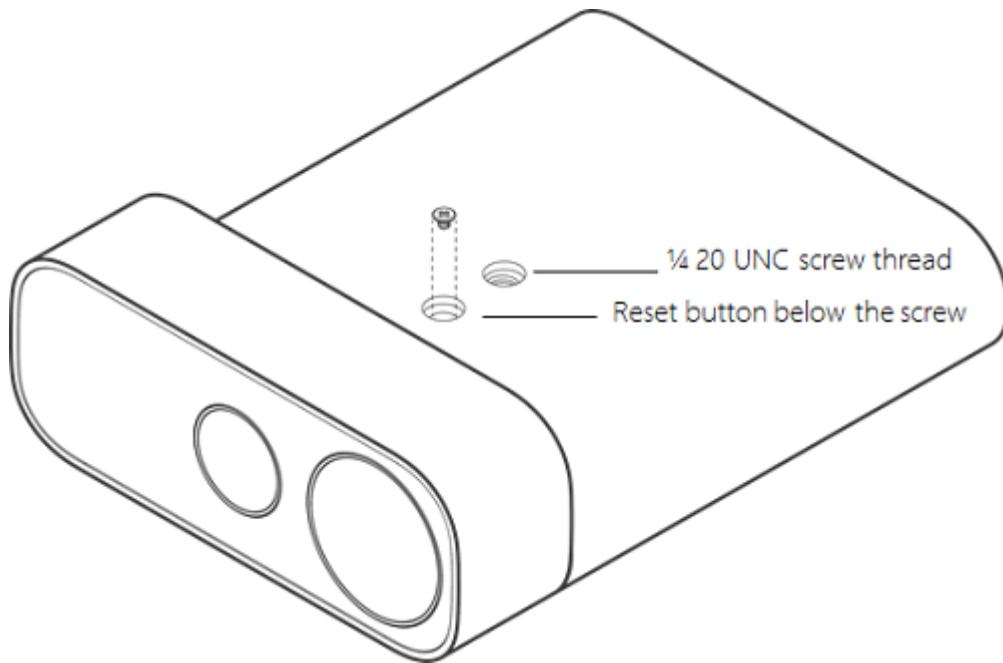
Azure Kinect DK peut consommer jusqu'à 5,9 W. Toutefois, la consommation dépend du cas d'utilisation.

Étalonnage

Azure Kinect DK est étalonné en usine. Les paramètres d'étalement des capteurs visuels et inertIELS peuvent être interrogés par programmation par le biais du SDK du capteur.

Récupération de l'appareil

Le microprogramme de l'appareil peut être réinitialisé vers le microprogramme d'origine à l'aide du bouton situé sous la vis.



Pour effectuer une récupération de l'appareil, consultez [ces instructions](#).

Étapes suivantes

- Utiliser le SDK du capteur Azure Kinect
- Configurer le matériel

Synchroniser plusieurs appareils Azure Kinect DK

Article • 01/06/2023

Chaque appareil Azure Kinect DK comprend des ports de synchronisation 3,5 mm (**Sync in** et **Sync out**) que vous pouvez utiliser pour lier plusieurs appareils. Une fois les appareils connectés, votre logiciel peut coordonner la synchronisation de leurs déclencheurs.

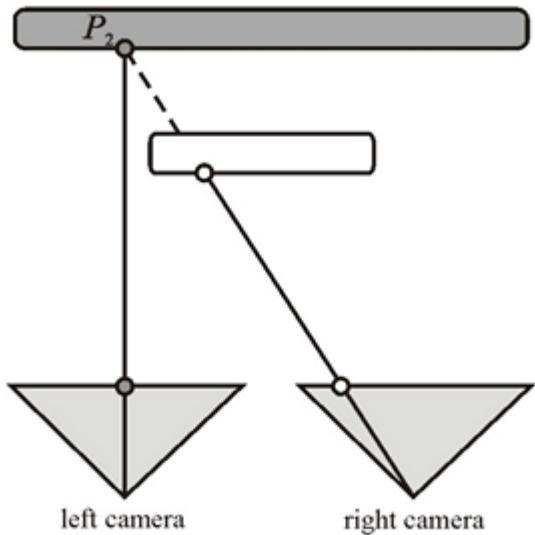
Cet article explique comment connecter et synchroniser les appareils.

Avantages de l'utilisation de plusieurs appareils Azure Kinect DK

Il existe de nombreuses raisons d'utiliser plusieurs appareils Azure Kinect DK, notamment les suivantes :

- Remplir les occlusions. Bien que les transformations de données d'Azure Kinect DK produisent une image unique, les deux caméras (profondeur et RVB) sont en fait légèrement écartées l'une de l'autre. Cet écart peut entraîner des occlusions. Une occlusion se produit quand un objet au premier plan bloque la vue d'une partie d'un objet à l'arrière-plan pour l'une des deux caméras d'un appareil. Dans l'image en couleurs obtenue, l'objet au premier plan semble projeter une ombre sur l'objet à l'arrière-plan.

Par exemple, dans le diagramme suivant, la caméra côté gauche voit le pixel gris « P2 ». Toutefois, l'objet de premier plan blanc bloque le faisceau IR de la caméra de droite. La caméra de droite ne reçoit pas de données pour « P2 ».



Des appareils synchronisés supplémentaires peuvent fournir les données occlues.

- Analyser des objets en trois dimensions.
- Augmenter la fréquence d'images effective en la réglant sur une valeur supérieure à 30 images par seconde (FPS).
- Capturer plusieurs images en couleur 4K de la même scène, toutes alignées dans les 100 microsecondes (μ s) par rapport au centre d'exposition.
- Augmenter la couverture de la caméra dans l'espace.

Planifier votre configuration de plusieurs appareils

Avant de commencer, veillez à consulter [Spécifications matérielles Azure Kinect DK](#) et [Caméra de profondeur DK Azure Kinect](#).

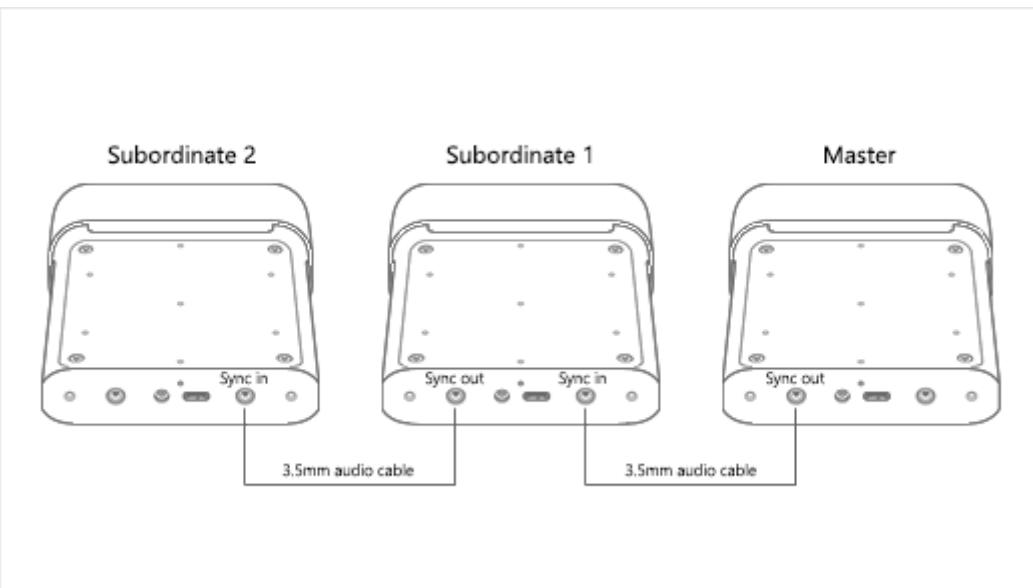
ⓘ Notes

Retirez le cache en plastique extérieur pour exposer les prises de synchronisation en entrée (« Sync In ») et de synchronisation en sortie (« Sync Out »).

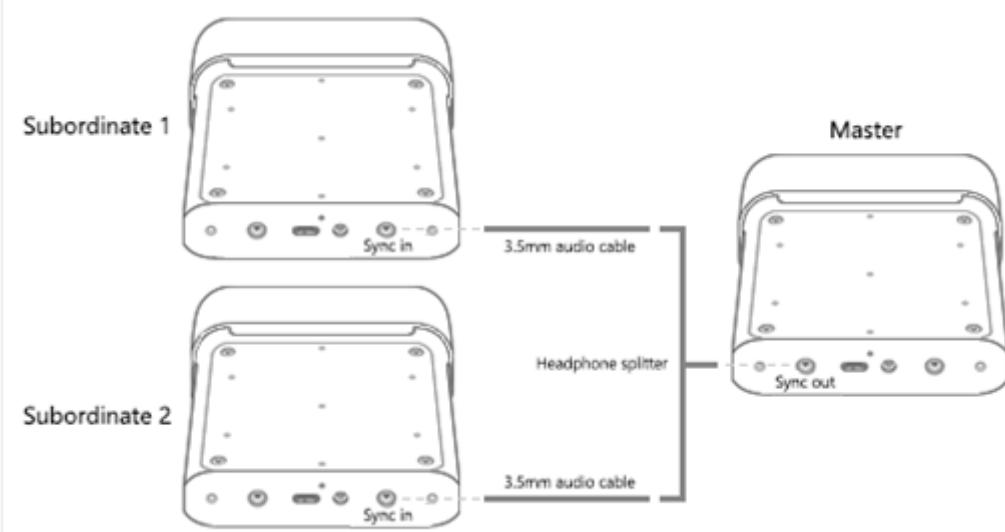
Sélectionner une configuration d'appareil

Pour la configuration de votre appareil, vous pouvez adopter l'une des approches suivantes :

- **Configuration en guirlande.** Permet de synchroniser un appareil maître avec jusqu'à huit appareils subordonnés.



- **Configuration en étoile.** Permet de synchroniser un appareil maître avec jusqu'à deux appareils subordonnés.



Utilisation d'un déclencheur de synchronisation externe

Dans les deux configurations, l'appareil maître fournit le signal de déclenchement pour les appareils subordonnés. Toutefois, vous pouvez utiliser une source externe personnalisée pour le déclencheur de synchronisation. Par exemple, vous pouvez utiliser cette option pour synchroniser des captures d'images avec d'autres équipements. Dans les configurations de connexion tant en guirlande qu'en étoile, la source du déclencheur externe se connecte à l'appareil maître.

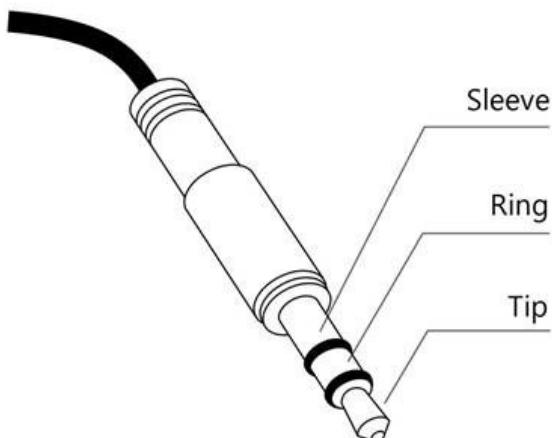
Votre source de déclencheur externe doit fonctionner de la même façon que l'appareil maître. Elle doit fournir un signal de synchronisation présentant les caractéristiques suivantes :

- Actif élevé
- Largeur d'impulsion : supérieure à 8 µs

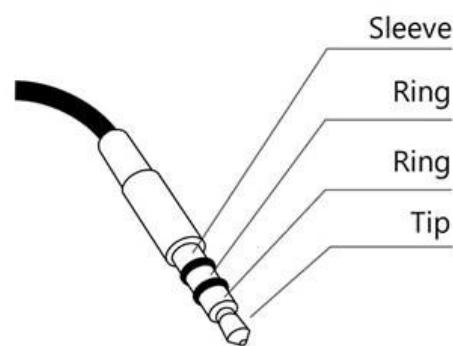
- TTL/CMOS 5V
- Capacité motrice maximale : pas inférieure à 8 milliampères (mA)
- Prise en charge de fréquence : Exactement 30 FPS, 15 FPS et 5 FPS (fréquence du signal VSYNC de la caméra couleur maître)

La source du déclencheur doit envoyer le signal au port **Sync in** de l'appareil maître en utilisant un câble audio 3,5 mm. Vous pouvez utiliser un câble stéréo ou mono.

L'appareil Azure Kinect DK met à la terre l'ensemble des gaines et anneaux du connecteur du câble audio. Comme illustré dans le diagramme suivant, l'appareil reçoit le signal de synchronisation uniquement de l'embout du connecteur.



TRS Audio Plug



TRRS Audio Plug

Pour plus d'informations sur l'utilisation de l'équipement externe, consultez [Utiliser un enregistreur Azure Kinect avec des appareils synchronisés externes](#)

Notes

Sync Out correspond à VSync pour la caméra RVB. Les timestamps de tous les appareils sont définis sur zéro et effectuent un comptage progressif. Microsoft n'a pas caractérisé la largeur minimale et maximale de l'impulsion de synchronisation et recommande d'imiter l'impulsion générée par la synchronisation en sortie d'un Azure Kinect DK.

Planifier les paramètres et la configuration logicielle de vos caméras

Pour plus d'informations sur la configuration de votre logiciel pour contrôler les caméras et utiliser les données d'image, consultez [SDK du capteur Kinect Azure](#).

Cette section aborde plusieurs facteurs qui affectent les appareils synchronisés (pas les appareils isolés). Votre logiciel doit tenir compte de ces facteurs.

Considérations relatives à l'exposition

Si vous souhaitez contrôler la synchronisation précise de chaque appareil, nous vous recommandons d'utiliser un paramétrage d'exposition manuelle. Avec le paramétrage d'exposition automatique, chaque caméra couleur peut modifier de façon dynamique l'exposition réelle. Étant donné que l'exposition affecte la synchronisation, ces modifications ont pour effet de désynchroniser rapidement les caméras.

Dans la boucle de capture d'images, évitez de définir plusieurs fois le même paramètre d'exposition. Nappelez l'API qu'une seule fois si nécessaire.

Éviter les interférences entre les plusieurs caméras de profondeur

Quand plusieurs caméras de profondeur acquièrent des images dont les champs de vision se chevauchent, chacune d'elles doit acquérir l'image associée à son propre laser. Pour éviter que les lasers interfèrent entre eux, les captures des caméras doivent être décalées entre elles d'au minimum 160 µs.

Pour chaque capture de caméra de profondeur, le laser s'active neuf fois et est actif pendant seulement 125 µs à chaque fois. Le laser reste ensuite inactif pendant 1 450 µs ou 2 390 µs, selon le mode de fonctionnement. Ce comportement signifie que le point de départ pour le calcul du décalage est de 125 µs.

De plus, les différences entre l'horloge de la caméra et celle du microprogramme de l'appareil augmentent le décalage minimal jusqu'à 160 µs. Pour calculer un décalage plus précis pour votre configuration, notez le mode de profondeur que vous utilisez et consultez le [tableau de synchronisation brute du capteur de profondeur](#). Les données de ce tableau vous permettent de calculer le décalage minimal (temps d'exposition de chaque caméra) à l'aide de l'équation suivante :

$$\text{Temps d'exposition} = (\text{Impulsions d'IR} \times \text{Largeur d'impulsion}) + (\text{Périodes d'inactivité} \times \text{Temps d'inactivité})$$

En utilisant un décalage de 160 µs, vous pouvez configurer jusqu'à neuf caméras de profondeur supplémentaires de façon à ce que chaque laser s'allume alors que les autres sont inactifs.

Dans votre logiciel, utilisez `depth_delay_off_color_usec` ou `subordinate_delay_off_master_usec` pour vous assurer que chaque laser IR s'exécute

dans sa fenêtre de 160 µs ou a un champ de vision différent.

ⓘ Notes

La largeur d'impulsion réelle est de 125 µs, mais nous indiquons 160 µs pour fournir une certaine marge de manœuvre. En utilisant NFOV UNBINNED comme exemple, chaque impulsion de 125 µs est suivie d'une inactivité de 1 450 µs. Le cumul de ces valeurs, (9 x 125) + (8 x 1450), fournit le temps d'exposition de 12,8 ms. La façon la plus proche d'entrelacer l'exposition de 2 appareils consiste à placer la première impulsion de la deuxième caméra dans la première période d'inactivité de la première caméra. Le retard entre la première caméra et la deuxième peut être réduit jusqu'à 125 µs (la largeur d'une impulsion), mais nous vous recommandons de ménager une certaine marge de manœuvre en utilisant 160 µs. Avec 160 µs, vous pouvez entrelacer les périodes d'exposition d'un maximum de 10 caméras.

Préparer vos appareils et autres composants matériels

En plus de plusieurs appareils Azure Kinect DK, il se peut que vous deviez obtenir des ordinateurs hôtes et d'autres composants matériels supplémentaires afin de prendre en charge la configuration que vous souhaitez créer. Utilisez les informations de cette section pour vous assurer que tous les appareils et composants matériels sont prêts avant de commencer la configuration.

Appareils Azure Kinect DK

Pour chaque appareil Azure Kinect DK à synchroniser, procédez comme suit :

- Assurez-vous que le microprogramme le plus récent est installé sur l'appareil. Pour plus d'informations sur la mise à jour de vos appareils, accédez à [Mettre à jour le microprogramme Azure Kinect DK](#).
- Retirez le cache de l'appareil pour voir les ports de synchronisation.
- Notez le numéro de série de chaque appareil. Vous l'utiliserez plus tard dans le processus d'installation.

Ordinateurs hôtes

En règle générale, chaque appareil Azure Kinect DK utilise son propre ordinateur hôte. Vous pouvez utiliser un contrôleur d'hôte dédié, en fonction de la façon dont vous utilisez l'appareil et de la quantité de données transférées via la connexion USB.

Assurez-vous que le Kit de développement logiciel (SDK) du capteur Azure Kinect est installé sur chaque ordinateur hôte. Pour plus d'informations sur l'installation du Kit de développement logiciel (SDK) du capteur, consultez le [Guide de démarrage rapide : Configurer Azure Kinect DK](#).

Ordinateurs Linux : mémoire USB sur Ubuntu

Par défaut, les ordinateurs hôtes basés sur Linux n'allouent au contrôleur USB que 16 Mo de mémoire du noyau pour gérer les transferts USB. Cette quantité est généralement suffisante pour prendre en charge un appareil Azure Kinect DK. En revanche, pour prendre en charge plusieurs appareils, le contrôleur USB doit avoir plus de mémoire. Pour augmenter la mémoire, procédez comme suit :

1. Modifiez `/etc/default/grub`.
2. Recherchez la ligne suivante :

```
Invite de commandes Windows  
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
```

Remplacez-la par cette ligne :

```
Invite de commandes Windows  
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash usbcore.usbfs_memory_mb=32"
```

ⓘ Notes

Ces commandes définissent la mémoire USB sur 32 Mo. Voici un exemple de paramétrage sur deux fois la valeur par défaut. Vous pouvez définir une valeur sensiblement plus élevée en fonction de votre solution.

3. Exécutez `sudo update-grub`.
4. Redémarrez l'ordinateur.

Câbles

Pour connecter les appareils entre eux et aux ordinateurs hôtes, vous devez utiliser des câbles 3,5 mm mâle-mâle (également appelés câbles audio 3,5 mm). Les câbles doivent

avoir une longueur inférieure à 10 mètres, et peuvent être stéréo ou mono.

Le nombre de câbles que vous devez avoir dépend du nombre d'appareils que vous utilisez, ainsi que de la configuration d'appareil spécifique. Le boîtier Azure Kinect DK n'inclut pas de câbles. Vous devez les acheter séparément.

Si vous connectez les appareils dans une configuration en étoile, vous devez également disposer d'un répartiteur de casque.

Connecter vos appareils

Pour connecter des appareils Azure Kinect DK dans une configuration de connexion en guirlande

1. Connectez chaque Azure Kinect DK à l'alimentation.
2. Connectez chaque appareil à son PC hôte.
3. Sélectionnez un appareil maître et branchez un câble audio 3,5 mm à son port **Sync out**.
4. Branchez l'autre extrémité du câble au port **Sync in** du premier appareil subordonné.
5. Pour connecter un autre appareil, branchez un autre câble au port **Sync out** du premier appareil subordonné, puis au port **Sync out** de l'appareil suivant.
6. Répétez l'étape précédente jusqu'à ce que tous les appareils soient connectés. Le dernier appareil ne doit avoir qu'une seule connexion câblée. Son port **Sync out** doit être libre.

Pour connecter des appareils Azure Kinect DK dans une configuration en étoile

1. Connectez chaque Azure Kinect DK à l'alimentation.
2. Connectez chaque appareil à son PC hôte.
3. Sélectionnez un appareil comme maître et branchez l'extrémité du répartiteur de casque à son port **Sync out**.
4. Connectez les câbles audio 3,5 mm aux extrémités « Split » du répartiteur de casque.
5. Branchez l'autre extrémité de chaque câble au port **Synch in** de l'un des appareils subordonnés.

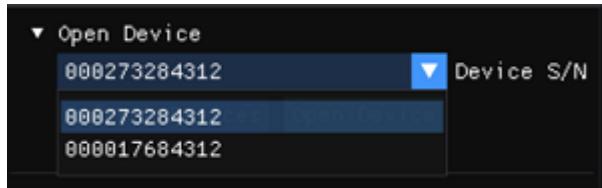
Vérifier que les appareils sont connectés et communiquent

Pour vérifier que les appareils sont connectés correctement, utilisez la [Visionneuse Kinect Azure](#). Répétez cette procédure si nécessaire pour tester chaque appareil subordonné combiné avec l'appareil maître.

ⓘ Important

Pour cette procédure, vous devez connaître le numéro de série de chaque Azure Kinect DK.

1. Ouvrez deux instances de la visionneuse Kinect Azure.
2. Sous **Ouvrir l'appareil**, sélectionnez le numéro de série de l'appareil subordonné à tester.



ⓘ Important

Pour obtenir un alignement précis de la capture d'images entre tous les appareils, vous devez démarrer l'appareil maître en dernier.

3. Sous Synchronisation externe, sélectionnez Sous.



4. Sélectionnez Démarrer.

ⓘ Notes

Étant donné qu'il s'agit d'un appareil subordonné, Azure Kinect Viewer n'affiche pas d'image après le démarrage de l'appareil. Aucune image n'est affichée tant que l'appareil subordonné n'a pas reçu de signal de synchronisation de l'appareil maître.

5. Une fois l'appareil subordonné démarré, utilisez l'autre instance de la Visionneuse Azure Kinect pour ouvrir l'appareil maître.
6. Sous Synchronisation externe, sélectionnez Maître.
7. Sélectionnez Démarrer.

Lors du démarrage de l'appareil Azure Kinect maître, les deux instances de la Visionneuse Azure Kinect doivent afficher des images.

Étalonner les appareils en tant qu'ensemble synchronisé

Une fois que vous avez vérifié que les appareils communiquent correctement, vous êtes prêt à les étalonner pour produire des images dans un domaine unique.

Dans un appareil, les caméras de profondeur et RVB sont calibrées en usine pour fonctionner ensemble. Cependant, lorsque plusieurs appareils doivent fonctionner ensemble, ils doivent être étalonnés pour déterminer comment transformer une image du domaine de la caméra qui l'a capturée dans le domaine de la caméra que vous souhaitez utiliser pour traiter les images.

Il existe plusieurs options pour l'étalonnage croisé d'appareils. Microsoft fournit l'[exemple de code GitHub green screen](#) qui utilise la méthode OpenCV. Le fichier LisezMoi de cet exemple de code fournit des détails et instructions supplémentaires pour l'étalonnage des appareils.

Pour plus d'informations sur l'étalonnage, consultez [Utiliser les fonctions d'étalonnage d'Azure Kinect](#).

Étapes suivantes

Après avoir configuré les appareils synchronisés, vous pouvez également apprendre à utiliser une

[API d'enregistrement et de lecture du Kit de développement logiciel \(SDK\) du capteur Azure Kinect](#)

Rubriques connexes

- [À propos du Kit de développement logiciel \(SDK\) du capteur Azure Kinect](#)
- [Spécifications matérielles pour Azure Kinect DK](#)
- [Démarrage rapide : Configurer Azure Kinect DK](#)
- [Mettre à jour le microprogramme Azure Kinect DK](#)
- [Réinitialiser Azure Kinect DK](#)
- [Visionneuse Azure Kinect](#)

Comparaison entre Azure Kinect et Kinect pour Windows v2

Article • 31/07/2023

Le matériel et les kits de développement logiciel (SDK) Azure Kinect DK présentent des différences par rapport à Kinect pour Windows v2. Les applications Kinect pour Windows v2 existantes ne fonctionnent pas directement avec Azure Kinect DK et nécessitent un portage vers le nouveau Kit de développement logiciel (SDK).

Matériel

Les grandes différences entre le kit de développement Azure Kinect et Kinect pour Windows v2 sont répertoriées dans le tableau suivant.

| Fonctionnalité | Type | Azure Kinect DK | Kinect pour Windows v2 |
|----------------------|-----------------|--|-------------------------------------|
| Audio | Détails | Tableau circulaire pour 7 micros | Tableau linéaire pour 4 micros |
| Capteur de mouvement | Détails | Accéléromètre sur 3 axes Gyroscope sur 3 axes | Accéléromètre sur 3 axes |
| Caméra RVB | Détails | 3 840 x 2 160 px @ 30 i/s | 1 920 x 1 080 px @ 30 i/s |
| Caméra de profondeur | Méthode | Temps de vol | Temps de vol |
| | Résolution | 640 x 576 px @ 30 i/s | 512 x 424 px @ 30 i/s |
| | | 512 x 512 px @ 30 i/s | |
| | | 1 024 x 1 024 px @ 15 i/s | |
| Connectivité | Données | USB 3.1 Gen 1 avec type USB-C | USB 3.1 Gen 1 |
| | Power | Bloc d'alimentation ou USB-C externes | Bloc d'alimentation externe |
| | Synchronization | Connexion appareil à appareil interne, externe, RVB & profondeur | Interne RVB & profondeur uniquement |
| Mécanique | Dimensions | 103 x 39 x 126mm | 249 x 66 x 67 mm |

| Fonctionnalité | Type | Azure Kinect DK | Kinect pour Windows v2 |
|----------------|------|---|------------------------|
| Masse | | 440 g | 970 g |
| Montage | | Un UNC ¼-20 Quatre points de vissage internes | Un UNC ¼-20 |

Pour plus d'informations, consultez le document [Matériel Azure Kinect DK](#).

Accès au capteur

Le tableau suivant fournit une comparaison des fonctionnalités d'accès aux capteurs.

| Fonctionnalité | Azure Kinect | Kinect pour Windows | Remarques |
|---|--------------|---------------------|--|
| Profondeur | ✓ | ✓ | |
| IR | ✓ | ✓ | |
| Color | ✓ | ✓ | Le format des couleurs prend en charge les différences, Azure Kinect DK prend en charge les contrôles de caméra suivants : exposition, balance des blancs, luminosité, contraste, saturation, netteté et réglage de gain |
| Audio | ✓ | ✓ | Les micros Azure Kinect DK sont accessibles via le Kit de développement logiciel (SDK) Speech ou une API native Windows |
| IMU | ✓ | | Azure Kinect DK offre une IMU sur 6 axes et Kinect pour Windows uniquement sur un 1 axe |
| Données d'étalonnage | ✓ | ✓ | Étalonnage du modèle de caméra compatible OpenCV |
| Synchronisation interne Profondeur-RVB | ✓ | ✓ | |
| Synchronisation externe | ✓ | | Azure Kinect DK autorise un délai programmable pour la synchronisation externe |
| Partager l'accès avec plusieurs clients | ✓ | | Le Kit de développement logiciel (SDK) du capteur Azure Kinect s'appuie sur WinUSB/libUSB pour accéder à l'appareil et n'a pas de service implémenté pour permettre le |

| Fonctionnalité | Azure Kinect | Kinect pour Windows | Remarques |
|---|--------------|---------------------|--|
| | | | partage d'accès à l'appareil entre plusieurs processus |
| Outil d'enregistrement/de lecture de flux | ✓ | ✓ | Azure Kinect DK utilise une implémentation basée sur un conteneur Matroska open source |

Fonctionnalités

Le jeu de fonctionnalités du Kit de développement logiciel (SDK) Azure Kinect est différent de Kinect pour Windows v2, comme indiqué ci-dessous :

| Fonctionnalité Kinect v2 | Type de données | Service/Kit de développement logiciel (SDK) |
|------------------------------|------------------|---|
| | Kinect v2 | Azure Kinect |
| Accès aux données du capteur | DepthFrame | Kit de développement logiciel (SDK) de capteur – Récupérer des images |
| | InfraredFrame | Kit de développement logiciel (SDK) de capteur – Récupérer des images |
| | ColorFrame | Kit de développement logiciel (SDK) de capteur – Récupérer des images |
| | AudioBeamFrame | Non prise en charge pour le moment |
| Suivi de corps | BodyFrame | Kit de développement logiciel (SDK) Body Tracking |
| | BodyIndexFrame | Kit de développement logiciel (SDK) Body Tracking |
| Mappage de coordonnées | CoordinateMapper | Kit de développement logiciel (SDK) de capteur – Transformations d'images |
| Suivi du visage | FaceFrame | Azure AI services : Visage ↗ |
| Reconnaissance vocale | N/A | Azure AI Speech ↗ |

Étapes suivantes

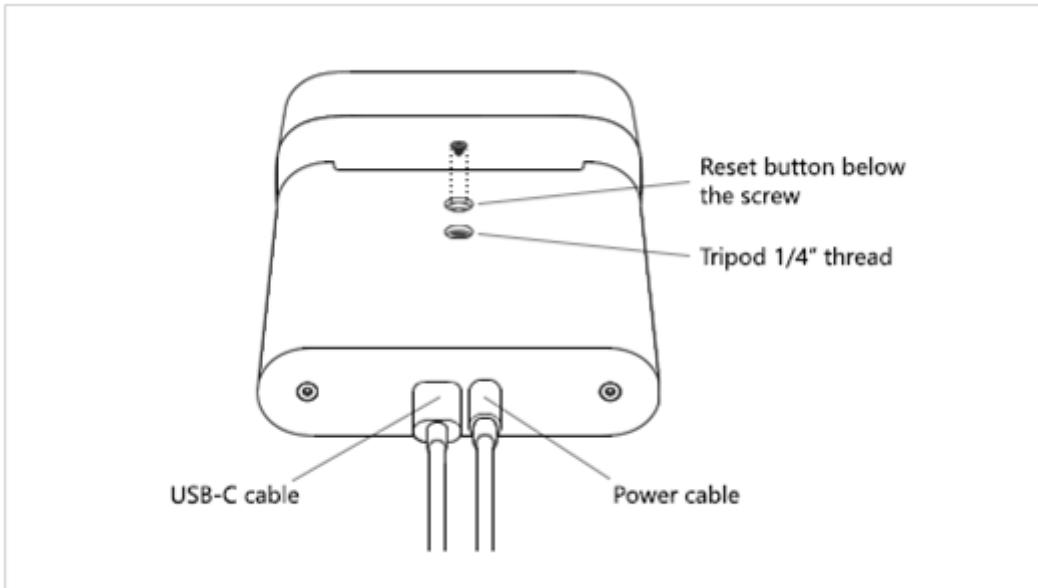
[Pages pour les développeurs Kinect pour Windows](#)

Réinitialiser Azure Kinect DK

Article • 01/06/2023

Vous pouvez être amené à réinitialiser Azure Kinect DK sur son image d'usine (par exemple, si une mise à jour du microprogramme n'a pas été installée correctement).

1. Mettez l'appareil Azure Kinect DK hors tension. Pour ce faire, retirez le câble USB et le câble d'alimentation.



2. Pour accéder au bouton de réinitialisation, retirez la vis située dans le verrou du trépied.
3. Rebranchez le câble d'alimentation.
4. Insérez le bout d'un trombone déplié dans le trou de la vis située dans le verrou du trépied.

⊗ Attention

N'utilisez jamais d'outil à bout pointu tel qu'une punaise pour appuyer sur le bouton de réinitialisation. Utilisez plutôt un outil à bout plat comme un trombone pour éviter d'endommager le bouton de réinitialisation.

5. Utilisez le trombone pour appuyer doucement sur le bouton de réinitialisation et le maintenir enfoncé.
6. Pendant que vous appuyez sur le bouton de réinitialisation, rebranchez le câble USB.

7. Au bout d'environ 3 secondes, le voyant d'alimentation devient orange. Une fois que le voyant a changé de couleur, relâchez le bouton de réinitialisation.

Une fois que vous avez relâché le bouton de réinitialisation, le voyant d'alimentation clignote en blanc et en orange pendant la réinitialisation de l'appareil.

8. Attendez que le voyant d'alimentation soit blanc et fixe.

9. Replacez la vis dans le verrou du trépied, par-dessus le bouton de réinitialisation.

10. Utilisez la visionneuse Azure Kinect pour vérifier que le microprogramme a été réinitialisé. Pour ce faire, lancez la [visionneuse Azure Kinect](#), puis sélectionnez **Device firmware version info** (Informations sur la version du microprogramme de l'appareil) pour voir la version du microprogramme qui est installée sur votre ordinateur Azure Kinect DK.

Vérifiez toujours que le microprogramme le plus récent est installé sur l'appareil. Pour connaître la dernière version du microprogramme, utilisez l'outil du microprogramme Azure Kinect. Pour savoir comment vérifier l'état du microprogramme, consultez [Vérifier la version du microprogramme de l'appareil](#).

Rubriques connexes

- [À propos d'Azure Kinect DK](#)
- [Configurer Azure Kinect DK](#)
- [Spécifications matérielles pour Azure Kinect DK : environnement d'exploitation](#)
- [Outil du microprogramme Azure Kinect](#)
- [Visionneuse Azure Kinect](#)
- [Synchronisation de plusieurs appareils Azure Kinect DK](#)

Options et ressources de support Azure Kinect

Article • 10/01/2024

Cet article vous présente les différentes options de support.

Support de la communauté pour les objets blob

Il existe plusieurs façons d'obtenir des réponses à vos questions via des forums publics :

- [StackOverflow](#), où vous pouvez poser des questions ou effectuer des recherches dans la bibliothèque de questions existante.
- [GitHub](#), où vous pouvez poser des questions, ouvrir de nouveaux bogues ou contribuer au développement du kit de développement logiciel (SDK) de capteur Kinect Azure.
- [Fournir des commentaires](#), où vous pouvez partager vos idées sur l'avenir du produit et voter pour une idée existante.

Support assisté

Il existe plusieurs façons d'obtenir un support assisté pour Azure Kinect.

Microsoft Q&A

Si vous souhaitez obtenir des réponses rapides et fiables à vos questions techniques sur les produits de la part d'ingénieurs Microsoft, de MVP (Most Valuable Professionals) Azure ou de notre communauté d'experts, contactez-nous sur [Microsoft Q&A](#), la destination favorite pour le support de la communauté Azure.

- [Microsoft Q&A pour Azure Kinect](#), où vous pouvez poser des questions ou effectuer des recherches dans la bibliothèque de questions existante.

Développement Azure Kinect sur Azure

Les abonnés Azure peuvent créer et gérer des demandes de support dans le portail Azure. Un support de développement individuel pour les services Body Tracking, Sensor

SDK, Speech device SDK ou Azure AI est disponible pour les abonnés Azure avec un [plan de Support Azure](#) associé à leur souscription.

- Un [plan de support Azure](#) est-il associé à votre abonnement Azure ? Connectez-vous au [Portail Azure](#) pour soumettre un incident.
- Vous avez besoin d'un abonnement Azure ? [Les options d'abonnement Azure](#) fournissent des informations supplémentaires sur les différentes options.
- Vous avez besoin d'un plan de support ? [Sélectionner un plan de support](#)

Azure Kinect en local ou autres services cloud

Pour obtenir un support technique en lien avec l'utilisation du Kit de développement logiciel (SDK) de capteur et du Kit de développement logiciel (SDK) de suivi de corps en local, ouvrez un ticket de support professionnel sur le [portail de support Microsoft](#).

Appareil Azure Kinect DK

Avant de contacter le support concernant le matériel, assurez-vous que vous avez configuré et mis à jour Azure Kinect DK. Pour tester si l'appareil fonctionne, utilisez la [visionneuse Azure Kinect](#). Pour en savoir plus, consultez notre page d'[aide sur Azure Kinect DK](#). Vous pouvez également consulter les [problèmes connus et leur résolution](#).

[Obtenez de l'aide](#) concernant une fonctionnalité d'appareil ou de capteur, des mises à jour de microprogramme ou des options d'achat.

Pour plus d'informations sur les offres de support, consultez [Support Microsoft pour l'entreprise](#).

Les déclarations de conformité UE pour les produits matériels Microsoft sont [ici](#).

Étapes suivantes

[Résolution des problèmes Azure Kinect](#)

Problèmes connus et résolution des problèmes liés à Azure Kinect

Article • 01/06/2023

Cette page décrit des problèmes connus et des conseils de dépannage relatifs à l'utilisation du Kit de développement logiciel (SDK) de capteur Azure Kinect DK. Pour des problèmes spécifiques du matériel du produit, consultez également les [pages du support technique](#).

Problèmes connus

- Problèmes de compatibilité avec les contrôleurs hôtes USB ASMedia (par exemple, circuit microprogrammé ASM1142) :
 - Certains boîtiers utilisant un pilote USB Microsoft peuvent se débloquer.
 - De nombreux PC ont également d'autres contrôleurs hôtes. Une modification du port USB3 peut alors être utile.

Pour d'autres problèmes liés au Kit de développement logiciel (SDK) de capteur, consultez [Problèmes GitHub ↗](#).

Collecte des journaux d'activité

La journalisation pour k4a.dll est activée via des variables d'environnement. Par défaut, la journalisation est envoyée à std::out et seules des erreurs et des messages critiques sont générés. Vous pouvez modifier ces paramètres de façon à ce que la journalisation accède à un fichier. Vous pouvez également ajuster la verbosité en fonction des besoins. Vous trouverez ci-dessous un exemple, pour Windows, de l'activation de la journalisation dans un fichier nommé k4a.log, qui capture des messages d'avertissement et de niveau supérieur.

1. `set K4A_ENABLE_LOG_TO_A_FILE=k4a.log`
2. `set K4A_LOG_LEVEL=w`
3. Exécutez le scénario à partir de l'invite de commandes (par exemple, lancez la visionneuse).
4. Accédez au fichier k4a.log et partagez-le.

Pour plus d'informations, consultez l'extrait du fichier d'en-tête ci-dessous :

Console

```

/**
 * environment variables
 * K4A_ENABLE_LOG_TO_A_FILE =
 *   0      - completely disable logging to a file
 *   log\custom.log - log all messages to the path and file specified - must
end in '.log' to
 *           be considered a valid entry
 *   ** When enabled this takes precedence over the value of
K4A_ENABLE_LOG_TO_STDOUT
*
* K4A_ENABLE_LOG_TO_STDOUT =
*   0      - disable logging to stdout
*   all else - log all messages to stdout
*
* K4A_LOG_LEVEL =
*   'c'  - log all messages of level 'critical' criticality
*   'e'  - log all messages of level 'error' or higher criticality
*   'w'  - log all messages of level 'warning' or higher criticality
*   'i'  - log all messages of level 'info' or higher criticality
*   't'  - log all messages of level 'trace' or higher criticality
*   DEFAULT - log all message of level 'error' or higher criticality
*/

```

La journalisation pour le Kit de développement logiciel (SDK) de suivi de corps K4ABT.dll est similaire, à ceci près que les utilisateurs doivent modifier un autre ensemble de noms de variables d'environnement :

Console

```

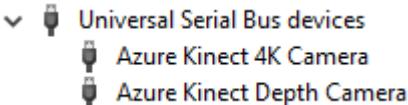
/**
 * environment variables
 * K4ABT_ENABLE_LOG_TO_A_FILE =
*   0      - completely disable logging to a file
*   log\custom.log - log all messages to the path and file specified - must
end in '.log' to
 *           be considered a valid entry
 *   ** When enabled this takes precedence over the value of
K4A_ENABLE_LOG_TO_STDOUT
*
* K4ABT_ENABLE_LOG_TO_STDOUT =
*   0      - disable logging to stdout
*   all else - log all messages to stdout
*
* K4ABT_LOG_LEVEL =
*   'c'  - log all messages of level 'critical' criticality
*   'e'  - log all messages of level 'error' or higher criticality
*   'w'  - log all messages of level 'warning' or higher criticality
*   'i'  - log all messages of level 'info' or higher criticality
*   't'  - log all messages of level 'trace' or higher criticality
*   DEFAULT - log all message of level 'error' or higher criticality
*/

```

La gestionnaire de périphériques n'énumère pas l'appareil

- Vérifiez le témoin d'état à l'arrière de l'appareil. S'il apparaît orange clignotant, cela signifie que vous avez un problème de connexion USB et que l'alimentation des insuffisante. Le câble d'alimentation doit être branché au transformateur fourni. Bien que le câble d'alimentation soit muni d'un connecteur USB de type A, l'appareil nécessite plus de puissance que ne peut en fournir un port USB de PC. Ne le connectez donc pas à un port de PC ou à un concentrateur USB.
- Vérifiez que le câble d'alimentation est connecté et que vous utilisez le port USB3 pour les données.
- Essayez de modifier le port USB3 pour la connexion de données (il est recommandé d'utiliser un port USB proche de la carte mère, par exemple, à l'arrière du PC).
- Vérifiez votre câble. Un câble endommagé ou de qualité médiocre peut entraîner une défaillance d'énumération (l'appareil continue de clignoter dans le gestionnaire de périphériques).
- Si vous êtes connecté à un ordinateur portable fonctionnant sur batterie, il se peut que celui-ci limite l'alimentation du port.
- Redémarrez le PC hôte.
- Si le problème persiste, il y a peut-être un problème de compatibilité.
- Si une défaillance s'est produite pendant la mise à jour du microprogramme et que l'appareil n'a pas récupéré de lui-même, effectuez une [réinitialisation aux paramètres d'usine ↗](#).

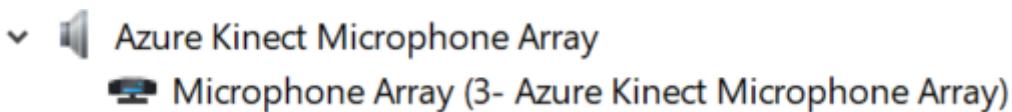
La visionneuse Azure Kinect ne s'ouvre pas

- Commencez par vérifier que le Gestionnaire de périphériques Windows énumère votre appareil.
- Vérifiez si vous disposez d'une autre application utilisant l'appareil (par exemple, application de caméra Windows). Une seule application à la fois peut accéder à l'appareil.
- Consultez les messages d'erreur dans le journal k4aviewer.err.
- Ouvrez l'application de caméra Windows et vérifiez que cela fonctionne.

- Redémarrez l'appareil, attendez que le témoin de diffusion en continu s'éteigne avant d'utiliser l'appareil.
- Redémarrez le PC hôte.
- Vérifiez que vous utilisez les pilotes graphiques les plus récents sur votre PC.
- Si vous utilisez votre propre build du Kit de développement logiciel (SDK), essayez d'utiliser la version officielle pour résoudre le problème.
- Si le problème persiste, [collectez les journaux](#) et formulez des commentaires.

Microphone introuvable

- Commencez par vérifier que le Gestionnaire de périphériques énumère le réseau de microphones.
- Si un appareil est énuméré et fonctionne correctement dans Windows, il se peut qu'après la mise à jour du microprogramme, Windows ait affecté un autre ID de conteneur à la caméra de profondeur.
- Vous pouvez essayer de réinitialiser celle-ci en accédant à Gestionnaire de périphériques, en cliquant avec le bouton droit sur « Réseau de microphones Azure Kinect », puis en sélectionnant « Désinstaller l'appareil ». Une fois cette opération terminée, détachez et rattachez le capteur.



- Après avoir redémarré la visionneuse Azure Kinect, réessayez.

Problèmes de mise à jour du microprogramme de l'appareil

- Si le numéro de version indiqué après la mise à jour est incorrect, il se peut que vous deviez redémarrer l'appareil.
- Une interruption de la mise à jour du microprogramme peut avoir dégradé son état et expliquer pourquoi elle n'est pas énumérée. Détachez et rattachez l'appareil, puis attendez 60 secondes pour voir s'il récupère. Si ce n'est pas le cas, effectuez une [réinitialisation aux paramètres d'usine ↗](#).

Problèmes de qualité d'image

- Démarrez la [visionneuse Kinect Azure](#) et vérifiez que l'appareil est positionné à l'abri de toute interférence, que le capteur n'est pas bloqué et que l'objectif est propre.
- Essayez différents modes de fonctionnement pour déterminer si le problème se produit dans un mode spécifique.
- Pour partager des problèmes de qualité d'image avec l'équipe, vous pouvez :
 1. Prendre un affichage de pause sur la [visionneuse Kinect Azure](#) et faire une capture d'écran ou
 2. Prendre un enregistrement effectué à l'aide de l'[enregistreur Azure Kinect](#), par exemple, `k4arecorder.exe -1 5 -r 5 output.mkv`

Horodatages d'appareil incohérents ou inattendus

L'appel de la fonction `k4a_device_set_color_control` peut entraîner des changements temporaires de synchronisation sur l'appareil qui peut prendre quelques captures pour se stabiliser. Évitez d'appeler l'API dans la boucle de capture d'images afin d'éviter la réinitialisation du calcul de synchronisation interne à chaque nouvelle image. Au lieu de cela, appelez l'API avant le démarrage de la caméra ou uniquement lorsque vous avez besoin de modifier la valeur dans la boucle de capture d'images. Évitez en particulier d'appeler la fonction

```
k4a_device_set_color_control(K4A_COLOR_CONTROL_AUTO_EXPOSURE_PRIORITY).
```

Compatibilité du contrôleur hôte USB3

Si le gestionnaire de périphériques n'énumère pas l'appareil, cela peut être dû au fait qu'il est connecté à un contrôleur USB3 non pris en charge.

Pour l'appareil Azure Kinect DK sur Windows, *les seuls contrôleurs hôtes pris en charge sont Intel, Texas Instruments (TI) et Renesas*. Le Kit de développement logiciel (SDK) Azure Kinect sous Windows s'appuie sur un ID de conteneur unifié. Celui-ci doit s'étendre aux périphériques USB 2.0 et 3.0 afin que le Kit de développement logiciel (SDK) puisse détecter les périphériques de profondeur, de couleur et audio qui se trouvent physiquement sur le même appareil. Sous Linux, un plus grand nombre de contrôleurs hôtes peuvent être pris en charge, car cette plateforme s'appuie moins sur l'ID de conteneur et davantage sur les numéros de série des appareils.

La question des contrôleurs hôtes USB se complique encore davantage quand plusieurs contrôleurs hôtes sont installés sur un PC. Lorsque les contrôleurs hôtes sont mélangés, l'utilisateur peut rencontrer des problèmes quand certains ports fonctionnent correctement et d'autres pas du tout. Selon la façon dont les ports sont connectés au boîtier, vous pouvez voir tous les ports frontaux qui rencontrent des problèmes avec Azure Kinect.

Windows : Pour déterminer le contrôleur hôte que vous avez, ouvrez le Gestionnaire de périphériques

1. Afficher -> Périphériques par type
2. Avec l'appareil Azure Kinect connecté, sélectionnez Caméras -> Caméra Azure Kinect 4K
3. Afficher -> Périphériques par connexion



Pour mieux comprendre quel port USB est connecté sur votre ordinateur, répétez ces étapes pour chaque port USB lorsque vous connectez l'appareil Azure Kinect DK à différents ports USB sur le PC.

La caméra de profondeur s'éteint automatiquement

Le laser que la caméra profondeur utilise pour calculer les données de profondeur d'image a une durée de vie limitée. Pour optimiser la durée de vie des lasers, la caméra de profondeur détecte quand les données de profondeur ne sont pas utilisées. La caméra de profondeur s'éteint lorsque l'appareil diffuse en continu pendant plusieurs minutes sans que l'ordinateur hôte lise les données. Cela a également une incidence sur la synchronisation de plusieurs appareils où les appareils subordonnés démarrent dans un état où la caméra de profondeur diffuse en continu et où les images de profondeur sont retenues en attendant que l'appareil maître commence à synchroniser les captures. Pour éviter ce problème dans des scénarios de capture avec plusieurs appareils, assurez-vous que l'appareil maître démarre dans la minute suivant le démarrage du premier appareil subordonné.

Utilisation du Kit de développement logiciel (SDK) de suivi de corps avec Unreal

Pour utiliser le Kit de développement logiciel (SDK) de suivi de corps avec Unreal, assurez-vous que vous avez ajouté `<SDK Installation Path>\tools` à la variable d'environnement `PATH` et copié `dnn_model_2_0.onnx` et `cudnn64_7.dll` sur `Program Files/Epic Games/UE_4.23/Engine/Binaries/Win64`.

Utilisation d'Azure Kinect sur un système Linux sans périphérique de contrôle

Le moteur de profondeur Azure Kinect sur Linux utilise OpenGL. OpenGL requiert une instance de fenêtre, laquelle requiert la connexion d'un moniteur au système. Solution de contournement pour ce problème :

1. Activez la connexion automatique pour le compte d'utilisateur que vous prévoyez d'utiliser. Reportez-vous à [cet article](#) pour obtenir des instructions sur l'activation de la connexion automatique.
2. Mettez le système hors tension, déconnectez le moniteur et mettez le système sous tension. La connexion automatique force la création d'une session x-server.
3. Connectez-vous via SSH et définissez la variable d'environnement DISPLAY `export DISPLAY=:0`
4. Démarrez votre application Azure Kinect.

L'utilitaire [xtrlock](#) permet de verrouiller immédiatement l'écran après la connexion automatique. Ajoutez la commande suivante à l'application de démarrage ou au service systemd :

```
bash -c "xtrlock -b"
```

Documentation C# manquante

La documentation C# du kit SDK du capteur est disponible [ici](#).

La documentation C# du kit SDK de suivi des corps est disponible [ici](#).

Modifications apportées au contenu des packages de suivi du corps

Les packages MSI et NuGet n'incluent plus les fichiers du package redistribuable Microsoft Visual C++. Téléchargez le dernier package [ici](#).

Le package NuGet est de retour mais n'inclut plus les fichiers Microsoft DirectML ou NVIDIA CUDA et TensorRT.

Étapes suivantes

[Autres informations de support](#)

Utiliser le Kit de développement logiciel (SDK) de capteur Azure Kinect pour enregistrer un format de fichier

Article • 01/06/2023

Pour enregistrer des données de capteur, le format de conteneur Matroska (.mkv) est utilisé, ce qui permet de stocker plusieurs pistes à l'aide de toute une série de codecs. Le fichier d'enregistrement contient des pistes pour le stockage des images en couleurs, de profondeur et IR, ainsi que des IMU.

Les détails de bas niveau du format de conteneur .mkv sont disponibles sur le [site Web Matroska ↗](#).

| Nom de piste | Format de codec |
|--------------------------------|--|
| COULEUR | Dépendant du mode (MJPEG, NV12 ou YUY2) |
| DEPTH | b16g (nuances de gris, big-endian 16 bit) |
| IR | b16g (nuances de gris, codage Big Endian 16 bite) |
| Unités de mesure inertielle | Structure personnalisée, consultez la structure d'échantillon d'IMU ci-dessous . |

Utilisation d'outils tiers

Vous pouvez utiliser des outils tels que `ffmpeg` ou la commande `mkvinfo` du kit de ressources [MKVToolNix ↗](#) pour afficher et extraire des informations de fichiers d'enregistrement.

Par exemple, la commande suivante extrait la piste de profondeur sous la forme d'une séquence de png 16 bits dans le même dossier :

```
ffmpeg -i output.mkv -map 0:1 -vsync 0 depth%04d.png
```

Le paramètre `-map 0:1` extrait la piste index 1 qui, pour la plupart des enregistrements, est celle de la profondeur. Si l'enregistrement ne contient pas de piste couleurs, il convient d'utiliser `-map 0:0`.

Le paramètre `-vsync 0` force ffmpeg à extraire les images telles quelles au lieu de tenter de se conformer à une cadence de 30, 15 ou 5 i/s.

Structure d'échantillon d'IMU

Si les données d'IMU sont extraites du fichier sans utilisation de l'API de lecture, elles ont une forme binaire. La structure des données d'IMU est présentée ci-dessous. Tous les champs sont codés en mode Little Endian.

| Champ | Type |
|--------------------------------------|----------|
| Horodatage de l'accéléromètre (μs) | uint64 |
| Données de l'accéléromètre (x, y, z) | float[3] |
| Horodatage du gyroscope (μs) | uint64 |
| Données du gyroscope (x, y, z) | float[3] |

Identification des pistes

Il peut être nécessaire d'identifier la piste Couleur, Profondeur, IR et ainsi de suite. L'identification des pistes est nécessaire lors de l'utilisation d'outils tiers pour lire un fichier Matroska. Les numéros de piste varient en fonction du mode de la caméra et de l'ensemble des pistes activées. Des balises sont utilisées pour identifier la signification de chaque piste.

Dans la liste ci-dessous, chaque balise est attachée à un élément Matroska spécifique et peut être utilisée pour rechercher la piste ou la pièce jointe correspondante.

Ces balises sont visibles avec des outils tels que `ffmpeg` et `mkvinfo`. La liste complète des balises figure dans la page [Enregistrement et lecture](#).

| Nom de la balise | Cible de la balise | Valeur de la balise |
|----------------------|---------------------------|--------------------------------|
| K4A_COLOR_TRACK | Piste Couleur | UID de piste Matroska |
| K4A_DEPTH_TRACK | Piste Profondeur | UID de piste Matroska |
| K4A_IR_TRACK | Piste IR | UID de piste Matroska |
| K4A_IMU_TRACK | Piste IMU | UID de piste Matroska |
| K4A_CALIBRATION_FILE | Pièce jointe d'étalonnage | Nom du fichier de pièce jointe |

Étapes suivantes

Enregistrement et lecture